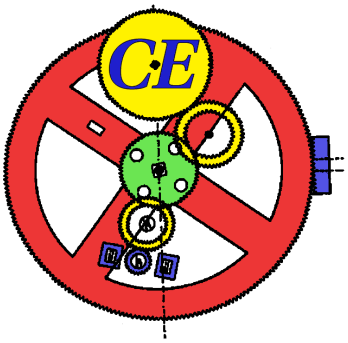


Computer Engineering
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

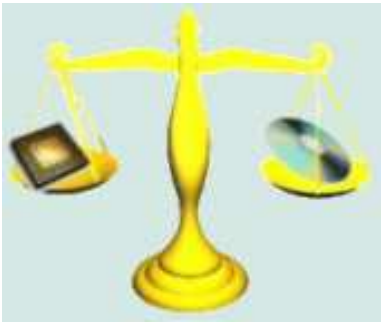
2011

MSc THESIS

A flexible high level modelling methodology for power and energy consumption

Omar Esli Jimenez Villarreal

Abstract



With the increasing complexity of current embedded applications, and the mobility required in embedded devices, new approaches are being proposed to optimize the power and the energy consumed by an application, at higher levels of abstraction levels and in earlier stages in a design flow. However, one essential part of a structured and guided optimization process, is the early prediction of the power and energy consumption using only the available information at early design stages. These predictions are used as function costs in optimization algorithms to prune the design space exploration in different design stages. In this thesis we aim to improve the partitioning process of the Delft Workbench design flow, for this purpose, we propose a modelling methodology that can generate power and energy models. The models can provide quantitative data that can be used to guide the decisions made in the partitioning process. The partitioning process in the DWB uses as level of abstraction a function described in a high level language (HLL), such as C-code, and targets heterogeneous architectures. Therefore, the methodology we propose can generate models that predict the power and energy consumed by a kernel when is running in a processing element of heterogeneous architectures, such as a general purpose processor (GPP) or an PPGA. For the validation of this methodology we designed a set of experiment that create models of power and energy consumption for a StrongARM processor (using the *Sim-Panalyzer* simulator), and a Virtex 5 FPGA (using the *xpwr* tool of Xilinx). A maximum absolute rooted mean squared error (RMSE) of 60mW was obtained for the power models, and a maximum absolute RMSE of 8.69×10^{-6} was obtained for the energy models.

CE-MS-2011-31

TU Delft
Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

A flexible high level modelling methodology for power and energy consumption

Generation of prediction models of energy and power consumption that require a HLL description as input

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

EMBEDDED SYSTEMS

by

Omar Esli Jimenez Villarreal
born in Oaxaca, Mexico

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

A flexible high level modelling methodology for power and energy consumption

by Omar Esli Jimenez Villarreal

Abstract

With the increasing complexity of current embedded applications, and the mobility required in embedded devices, new approaches are being proposed to optimize the power and the energy consumed by an application, at higher levels of abstraction levels and in earlier stages in a design flow. However, one essential part of a structured and guided optimization process, is the early prediction of the power and energy consumption using only the available information at early design stages. These predictions are used as function costs in optimization algorithms to prune the design space exploration in different design stages. In this thesis we aim to improve the partitioning process of the Delft Workbench design flow, for this purpose, we propose a modelling methodology that can generate power and energy models. The models can provide quantitative data that can be used to guide the decisions made in the partitioning process. The partitioning process in the DWB uses as level of abstraction a function described in a high level language (HLL), such as C-code, and targets heterogeneous architectures. Therefore, the methodology we propose can generate models that predict the power and energy consumed by a kernel when is running in a processing element of heterogeneous architectures, such as a general purpose processor (GPP) or an PPGA. For the validation of this methodology we designed a set of experiment that create models of power and energy consumption for a StrongARM processor (using the *Sim-Panalyzer* simulator), and a Virtex 5 FPGA (using the *xpwr* tool of Xilinx). A maximum absolute rooted mean squared error (RMSE) of 60mW was obtained for the power models, and a maximum absolute RMSE of 8.69×10^{-6} was obtained for the energy models.

Laboratory : Computer Engineering
Codenumber : CE-MS-2011-31

Committee Members :

Advisor:	K.L.M. Bertels, CE, TU Delft
Chairperson:	K.L.M. Bertels, CE, TU Delft
Member:	K.L.M. Bertels, CE, TU Delft
Member:	G.K. Kuzmanov, CE, TU Delft
Member:	Dr.ir. T.G.R.M. van Leuken, ME, TUDelft

Esta tésis la dedico con todo mi cariño a mi familia, especialmente a mis padres, hermanos y mi sobrino. Gracias al cariño, apoyo y libertad que siempre me han brindado, tengo el orgullo y satisfaccin de terminar mi maestría en Embedded Systems en TUDelft con este trabajo de investigación. También dedico este trabajo a mis supervisores de tésis y a todos mis amigos, en México y Delft, y les agradezco por su amistad y apoyo!

I dedicate this work to my family, especially to my parents, my sister, my brother, and my nephew. Thanks for the love, support and freedom you have given me, I'm proud and satisfied now that I finish my MSc in Embedded Systems in TUDelft. I also dedicate this work to my thesis supervisors, and to all my friends (in México and Delft).

Contents

List of Figures

List of Tables

Acknowledgements

I want to express my grattitude to my thesis supervisors: Dr. Koen Bertels for supervising my work and for his inspiration, to Roel Meeuws and Arash Ostadzadeh for the endless hours they have spent supervising and revising my work. Furthermore, I want to thank Roel Meeuws for his work in Quipu, which is the base of this work, and to Arash Ostadzadeh for his work in Quad, whose work is used in this thesis. Thanks to Dr. Anca Molnos for all her support and for taking the time to review Chapter 3 of my thesis report. I would like to thank to the members of my thesis committee, Dr. G.K. Kuzmanov and Dr.ir. T.G.R.M. van Leuken, for the time spend reading and assesing my thesis research. And special thanks to all the Computer Engineering group who were always helpful and provided me support and guide during this thesis work.

Omar Esli Jimenez Villarreal
Delft, The Netherlands
October 28, 2011

1 Introduction

Nowadays, there are many factors that increase the ammount of power consumption in Embedded systems. Among the most important we can list: technology scaling, the rise of operational frequency and the increasing complexity of Embedded systems. Power consumption has been an important research topic because of the negative effects induced in Embedded systems. We can divide these negatives effects as follows:

- *Temperature-related problems*, where more complex and expensive cooling systems are required to reduce heating of chips, in order to avoid damage and the degradation of performance in the system.
- *Reduction of battery life in mobile systems.* Battery life is impacted by the energy consumption of internal components of an Embedded system.

These effects have increased because of the growth of power consumption in current applications. Thus the reduction of power consumption has becoming a key design challenge.

This problem has been tackled at different abstraction levels. At the lowest abstraction level, the design of computer realizations has focused on the reduction of power consumption by reducing sillicon area, optimizing the geometry of transistors, decreasing voltage of the power source required by the designs, and others optimizations that aim to reduce the power required by IC designs. On higher levels, the design of computer implementations have presented an oportunity to reduce power consumption by optimizing clock-trees architecture or using algorithmic optimization and RTL transformations. However, these optimizations have addressed reduction of power consumption in general Computer commodities used in customized Embedded Architectures. But with the ever-increasing complexity of these systems it is required to reduce power consumption in a higher level of abstraction, while reducing design efforts and time to market (TTM).

Recent work has been done to reduce power consumption at System design level (SDL). Nevertheless one of the main problems at SDL is that it is difficul to determine the effect of decisions made at this level, on the eventual power consumption. Usually, the hardware is not available yet, or the required time to physically measure the effects on power consumption is too time-consuming, so that it is not feasible to try multiple design options to find the optimal choice.

This problem requires a faster method to predict the power consumption caused by System design decisions. The methods currently used to predict power consumption rely on power models that abstract away from unnecessary details, in order to reduce the prediction time at the cost of a reduced accuracy.

At SDL, different prediction methodologies have been proposed, that use power models of lower abstraction levels to predict power consumption. Although these methodologies are accurate, the complexity of modern systems increase the prediction time. As

a result, we propose in this thesis a modelling methodology that can be used to generate power and energy models of functions running in processing elements, such as a GPP or FPGA. These models can be used at SDL, because the models require as input the HLL (C-code) description of a function.

Finally, since the ultimate goal is to find the most optimal system design that fits the required specifications (including power consumption), it is important to introduce the prediction models in the design flow of Embedded Systems.

1.1 Need of a power-aware design flow

In recent days, the heterogeneous architectures have become more and more common in embedded systems. They are composed of computer commodities like General Purpose Processors (GPP), Digital Signal Processors (DSP), or, recently, the usage of Field Programmable Gate Arrays (FPGA), along with memory blocks arranged in different hierarchies, types, and pheripherals. These architectures acknowledge the basic characteristic of Embedded systems: its composition. Embedded systems are formed by hardware and software modules. In the design of Embedded systems the right combination of hardware and software has to be chosen, which results in the most efficient product that meets the specifications required. This type of design is called HW/SW co-design, as explained in [20], which takes into account the behavioral specification of the product and also the available components. This design encourages reuse which is very important when coping with the increasing complexity of new Embedded systems and stringent time-to-market requirements.

The HW/SW co-design is an structured design flow that aims to optimize different design metrics, such as, performance, or cost. The structured flow, provides the opportunity to use optimization algorithms in order to improve the design metrics of an embedded system.

Therefore, in order to reduce the power consumption of Embedded systems a power aware design flow has to be followed, which can led to products that meet the specifications, using less power. In [20], we find a simplified design information flow for embedded systems. The diagram is depicted in Figure 1.1. This design flow starts with the idea of a product to be designed. This idea is captured in a formal way through a design specification. Once a specification is available, designers start an iterative process to implement the specification until a final product that meets the requirements is obtained.

In the design flow presented in Figure 1.1, a series of activities is depicted that are carried out during the development process of an Embedded system. These activities take place after a specification is available and include:

- *Task level concurrency management.* During this activity, the tasks that will be present in the final system are identified.
- *High-level transformations.* This activity involves the application of transformations in the specification (assuming that part of the specification is depicted as a C sequential code). These transformation aim to optimize certain design metrics, like performance.

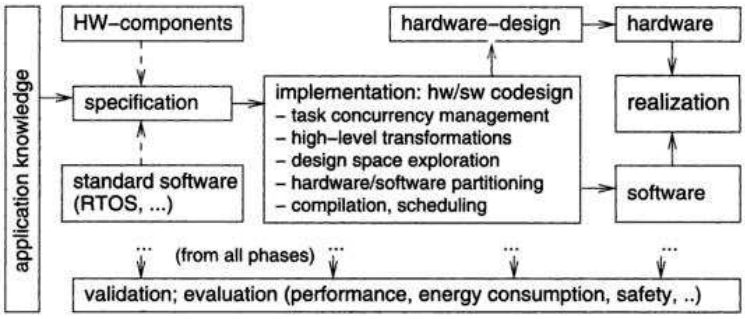


Figure 1.1: Simplified design information flow as presented in [20]

- *Hardware/Software partitioning.* During this activity the tasks identified are mapped to either hardware or software.
- *Compilation.* This activity includes the compilation of tasks mapped to software.
- *Scheduling.* This activity is performed in several stages of the design, for instance: HW/SW partitioning, task level concurrency management. And involves setting star-time to the tasks.
- *Design space exploration* Usually different designs meet the requirements of a system, thus a selection of one design is made during this stage.

The previously presented activities can have different orderings. The order in which these activities are performed constitutes a design flow also called design methodology.

In [13], a review of design methodologies used in recent years was presented. This review is presented as follows:

- *Bottom-Top.* This methodology is based on an intuitive approach of building blocks before the final product is assembled. In this methodology designers start from the lower abstraction level and build modules that will be stored in libraries. These libraries will be used later in the design of more modules in a higher abstraction layer.
- *Top-Bottom.* In this methodology, the layout of the entire design is made before the structure of the components that constitute the Model of Computation (MoC) is defined. After the layout of components in the higher level of abstraction is defined, a refinement process is started in the next lower level of abstraction to define the structure of the modules. This process continues until all the structures of each modules is defined.
- *Meet-in-the-middle.* This methodology takes advantage of Bottom-Top and Top-Bottom methodologies by starting the design with a MoC but using a top-down methodology for higher abstraction levels and bottom-top methodology for lower abstraction levels. The starting abstraction level is based on the current available

CAD tools. Where in lower abstraction levels these CAD tools are mature and well known by designers, in higher abstraction levels they do not exist or are still under development.

- *Platform Methodology.* This methodology relies on predetermined platforms defined by well-known platforms suppliers or in-house developed platforms. These platforms are composed of some standard components arranged in well-defined layouts and are used as the starting point of a design. These platforms are further customized, by adding customized components which are designed in the lower abstraction level. These customized components are built as standard cells which are used to modify the layout of higher level components. The predefined layout is modified using customized components to produce a final layout for the design.

For this research thesis the design flow and methodology used are defined in the Delft Workbench (DWB) project [5]. Further details of DWB are presented in Chapter 2. Therefore, this thesis aims to improve the partitioning process in the HW/SW co-design flow of DWB project. The goal is to predict the power and energy consumed by a task when it is mapped to a processing unit, such as, a GPP or an FPGA. Afterwards, the predictions are used to optimize the HW/SW partitioning process.

1.2 Scope of this research

The scope of this thesis research can be explained as follows:

- Analyze the characteristics of the HW/SW partitiong process in the DWB design flow.
- Based on the analysis, derive a modelling approach which can be used to build power and energy models of processing units in heterogenous architectures.
- Implement the modelling methodology to build an energy and a power model for two processing elements in heterogeneous architectures, the GPP and the FPGA.
- Validate the accuracy of the models.

1.3 Research questions

In order to achieve the proposed goal of this thesis, it is important to answer the following questions during the course of this thesis research:

- *Would an automatic high level modelling methodology help to investigate the different parameters that affect power consumption?*
- *Are the low level details of a HW processing unit necessary during the modelling process? Does considering the HW processing unit as a black-box provides more flexibility and scalability to the methodology?*

- *Does using a HLL (C code) as input for a prediction model reduces the prediction time of the model? Do the predictions obtained with this model provide qualitative data useful for the partitioning process in the DWB design flow?*

1.4 Overview

The rest of the thesis is organized as follows. Power consumption analysis, prediction and optimization have been addressed by the research community in several works. Therefore, Chapter 2 presents a review of the relevant work within the scope of this thesis and describes the motivation of this research work. In Chapter 3 we present the modelling methodology proposed in this thesis, the rationale of the methodology is presented as well. In order to validate this methodology, we designed a set of experiments. The experimental setup of these experiments is also presented in Chapter 3. Finally, in Chapter 4 the analysis of the experimental results are presented, along with the results of the modelling process that generates the models of power and energy. In Chapter 4, we also present the validation method to validate the accuracy of the models generated with the methodology. Finally Chapter 5 presents the conclusions and suggestions for future work of this thesis research.

2 Related Research

The problem of reducing power consumption, in the design of embedded systems, has been exhaustively investigated in the past decades. The problem has been addressed at different levels of abstraction:

- Transistor level
- Register Transfer Level (RTL)
- Algorithmic level
- System level

The solutions to this problem include the optimization of hardware and software components, as well as methodologies to reduce power consumption in the design of embedded systems. In this thesis we are proposing a solution to evaluate power and energy consumption during HW/SW partition process. This evaluation can be used to optimize energy consumption and control the maximum power consumed by the complete design. Hence it is important to present the related work and background of this thesis work.

As explained in the previous chapter, the HW/SW partitioning process is one of the earliest design stages of HW/SW co-design. *During this design stage, the tasks to be present in the final system are mapped to processing units.* A processing unit can be a General Purpose Processor (GPP), a special purpose processor like a Digital Signal Processor (DSP) , or an acceleration unit like a Field Programmable Gate Array (FPGA). The work carried out in this thesis is made in the context of the Delft Workbench (DWB) project. Therefore, we narrow down this survey to two components used in heterogeneous systems, GPPs and FPGAs.

This chapter is organized as follows, Section 2.1 presents the previous modelling techniques that predict power consumption in FPGAs at different abstraction levels. Section 2.2 presents a survey of the existing prediction methodologies to predict power and energy in GPPs. Section 2.3 presents the background of this thesis, which provides the contextual information for this work. Finally section 2.4 presents a conclusion of the survey presented in this chapter.

2.1 FPGA

Currently different types of accelerators exist in heterogeneous architectures. These accelerators are designed to improve the performance of the entire system by accelerating tasks which are frequently executed. Some accelerators can provide a high performance for specific applications, but suffer performance degradation with other applications.

This degradation is caused by the low flexibility of the ASIC design used to build these accelerators.

Among the accelerators currently available, the FPGA offers the higher flexibility compared with ASIC-based accelerators. And although the performance of a FPGA is still lower compared with ASIC designs, the design time is shorter. The popularity of FPGAs has increased, especially with the improvement of FPGA fabrics and technology scaling. FPGA-based designs have even been introduced in mobile embedded designs although with low and medium production ranges. However, the FPGAs have not been introduced in embedded systems produced in large scale, because, their power consumption is still higer compared to ASIC design.

In order to reduce the power consumption of heterogeneous systems containing FPGAs, it is important to have models which predict power consumption that can be used to optimize power consumption of a design. Such modelling efforts for FPGAs at different level of abstraction have been proposed earlier. In this section, recent works in modelling power consumption in a FPGA are presented. The remainder of this section is organized as follows:

- **FPGA-micro architecture Power Modelling.** Presents the research made to model power consumption at low abstraction level (transistor level).
- **Low-level Power Modelling.** This section presents modelling techniques for power consumption at RTL level. The difference with subsection 2.1.3 is that the models presented in this section require as input low abstraction-level information to predict power consumption.
- **High-level Power Modeling.** Presents modelling techniques used to optimize FPGA-based designs at system level or algorithmic level.

2.1.1 FPGA-micro architecture Power Modeling

A general approach to reduce power consumption of FPGA-based designs is to model and optimize the power consumed by the internal components of an FPGA. Using this approach any future FPGA-based design is improved. In this subsection, we present previous research that follows this general approach. The models presented here are used to optimize the micro-architecture of FPGAs.

We have to notice that the modelling techniques used during ASIC design can also be used during FPGA micro-architecture design, because the design is conducted at transistor level. Nevertheless, there are other modelling techniques, which take into account specific characteristics of FPGAs to model power consumption. Therefore in this subsection we only present the modelling techniques fully influenced by FPGA characteristics and leave the ASIC modelling techniques out.

A specific way to predict power consumption is to use a known model of a lower abstraction level and then through simulation of a design, obtain the power consumed by the design. This approach is followed by Poon et.al. in [31]. A power model that estimates dynamic, short circuit, and leakage power is presented in [31]. This model targets island-type FPGAs with logic blocks, switch blocks, connection blocks, routing elements

and an H-tree network. The work proposed was integrated in the VPR Computer Aided Design (CAD) tool. The additions made to VPR include an activity estimator module and a transistor level power model applied to each component of the FPGA. The Activity estimator tool calculates switching activity using transition density of a signal (calculated per Look-Up Table(LUT)). Then a transistor level model for each element of the FPGA is used (clock-tree, flip-flops, input mux, LUT and routing resources) along with the calculated transition density, to calculate dynamic power consumption. The short-circuit power is assumed to be 10% of dynamic power. And a leakage power model at transistor level is used to obtain the static power consumption. The models presented in [31] were validated against HSPICE, but no exact information was provided about the error of the models. Nevertheless, the authors state that a significant absolute error can be found using this model, but that it is still accurate enough to evaluate architectural trade-offs in FPGA design, and assessment of efficiency of CAD tools. Although, this approach provides information of power consumption, it usually involves high simulation times and requires detailed information of the design implementation.

Another approach is used in [11] based on existing models, which can be used to predict the power consumption of each component in a FPGA. Then those predictions are scaled, based on the resource utilization of each of those components. Concretely [11] presents a pre-silicon dynamic power estimation methodology. The methodology is applied to a coarse-grained FPGA architectural model, specifically to Spartan-3 FPGA. The programmable fabrics, routing elements, and clock distribution mechanism are resources considered in this prediction methodology. The first step in the methodology consists of the characterization of each resource using simulation tools (HSPICE or Nanosim) to find capacitance of each block. Then the dynamic power consumption is predicted, based on resource utilization of a design, switching activity (both are obtained from output files of after route and place in ISE), and the characterized block capacitance. The accuracy of the prediction methodology was compared against silicon measurements. It reports an average error of 18% with a max of 27%.

An evaluation in terms of energy and power consumption of bi-directional and unidirectional FPGA routing architectures is presented in [16]. Although this work doesn't present a modelling technique itself, it performs an analysis of FPGA components which are integrated into a CAD tool which predicts power consumption. The work presents general observations on how these two types of architectures affect speed, area, and power consumption. Also, it describes the effect on the critical path delay, power, and energy consumption of FPGAs, caused by the buffer size of routing elements. From the results obtained the authors concluded that a unidirectional architecture performs better in terms of area, performance, and energy consumption under most of the cases. However, bi-directional architectures consume less energy at the cost of an increase of area when the operating frequency is between any values in KHz to 10 MHz. The results of this work have been integrated in VPR5.0 a power estimation framework created by [31].

The work presented in this subsection provides a flexible way to predict power consumption of an FPGA component. The main advantage is that it can be adapted for different FPGAs architectures. However, the power models and the energy models, require as input a detailed transistor-level design description, which is not available in

earlier stages of design.

2.1.2 Low-level Power Modeling

In this subsection we present related models and techniques that aim to optimize power consumption of FPGAs during architectural design. The difference between the techniques lies in the prediction model used. Each model uses different input parameters of the components used in a design. The granularity of the FPGA used to build the models is also different. The main characteristic of the models presented in this section is that the power prediction is made using architectural design parameters and statistical signal metrics. However, since the signal metrics are obtained from a detailed description of the circuit implementation, these models cannot be used in architectural design without having transistor level design information of the design.

One model for power consumption prediction in FPGA-based design is presented in [34]. The model uses input/output signal statistics (average signal probability, input signal transition density, input signal spatial correlation, output signal transition density) to relate dynamic power consumption in a design. The model is built using a set of input signal samples, which are simulated in a timing simulator, to obtain input/output signal statistics. Since this model is built from input samples, not all possible input samples can be used for training, thus a statistical regression method is used to cope with this problem. Results show that this model has a similar error when input traces of a design are completely different to input samples used to train the model. The average relative error of this model is 3.1% for completely different input signals, and 1.7% with similar input signals.

In [17] a model of power dissipation at RTL level is presented. The power macro-model presents an equation to relate power consumption with each operator found in a Hardware Description Language (HDL) file (like adder, subtractor, multiplier, divider, or logic operations like AND, OR, etc.). It uses input design metrics like average input transition density, average input spatial correlation and input bit width. Thus this model takes into account internal configuration of a module plus input signal statistics. The accuracy of the macro-model is validated with XPower tool available in the ISE suite of Xilinx. And an average error of 3.14% is reported.

In [27] presents a power prediction tool developed in Java. The tool aims to calculate power consumption of a design implemented in a FPGA. The prediction is made using two inputs files. One file is a configuration file produced by FPGA CAD tools, which describes the configuration and connections of the Configuration Logic Blocks(CLBs) for a determined custom computing design implemented in an FPGA. The other file is an input signal activity file, where all input signals connected to a custom computing design are associated to probabilistic parameters that characterize each input signal. The equation that models power consumption per signal uses the distance between CLBs, the capacitance of resources, the voltage level of the FPGA, the frequency, and the activity density of the signal. In the previous equation the capacitance is unknown, therefore a set of test-benches implemented in an FPGA were used to derive the capacitance of the different elements in the design. The test-benches used to calibrate the tool were two Finite Impulse Response (FIR) filters. An average prediction error of 5% was reported

for designs similar to the test-benches, however, a larger error (less than 10%) appeared for designs which were not similar to the filters used for calibration. The accuracy of this tool was compared with real on-chip measurements.

The techniques presented in this section provide high accuracy in the prediction. The drawback is that the design-time of a system increases because the models require input/output signal statistics. Getting the signal statistics involves transition simulation of the design. For instance the signal activity of a design, can be provided as output files generated by FPGA CAD tools. Therefore, an early design space exploration of an architecture using these techniques would be really slow. This problem makes these techniques nearly impossible to be used for power prediction in early design stages, such as HW/SW partitioning.

2.1.3 High-level abstraction Power Modeling

In this subsection, recent work that proposes power optimization in FPGA-based designs at high-abstraction levels, like architectural or algorithmic design level, is presented. The main characteristics of the research presented are:

- The power prediction does not require statistical information about input/output signals of a design.
- Functional blocks in the FPGA-based design are identified, then the power consumed by each block is characterized during the modelling phase. Only blocks to be implemented in an FPGA are considered by these models.

The models of each functional block implemented in the FPGA are used to optimize power consumption of the entire design. Consequently, the design optimization process can be conducted faster when it is compared with the techniques presented in subsections 2.1.1 and 2.1.2.

A review of design optimizations at high-abstraction-level for FPGA-based design is presented in [8]. In this work a classification of design optimization techniques is done deriving two main areas for optimization.

- System Level techniques with algorithmic and behavioral transformations.
- Architectural Level techniques using parallelism and pipelining.

At System Level, Distributed arithmetic is identified as an algorithmic optimization technique which reduces area at expense of circuit complexity. However, overall reduction in power consumption is achieved. Parallelism and pipelining are identified as techniques to reduce power consumption at Architectural Level. These techniques reduce the operational frequency at the expense of increased area. Finally, Functional Level Power Analysis and Modeling (FLPAM) are proposed as a novel approach to model power consumption.

In [9] a High-Level (System Level) model to predict power consumption of FPGA-based designs is presented. The model is integrated in a design flow to allow an iterative power optimization process. The model was build following these stages:

1. Build a power chart by measuring power for each individual component that affects dynamic power, i.e. signal, logic, clock freq, I/O, etc.

2. Identify system variables that affect a design: frequency, area, vector length, supply voltage.

3. Choose a mathematical model for each of the components found in 1.

4. Derive coefficients for each model that relates system variables identified in 2 with the individual components that affect dynamic power. The coefficients are derived from power charts that contained the measured data in step 1.

5. Optimize model in an iterative process looking for convergence of coefficients.

6. Determine optimal function parameters to build the final model.

With this model system variables like frequency, area, vector length, and power supply can be used at system level design to predict power consumption of a design. The methodology presented to derive a model can be followed again if a change in platform is required. The accuracy of the model is between 92% and 100%, considering each model separately. However, it is not specified what validation methodology was used for each model.

For more complex designs implemented in an FPGA, like a soft-processor, the work presented in [43] proposes a hybrid power model to predict the power consumed by an application running on the soft-processor. The hybrid model is integrated by Functional Level Power Analysis (FLPA) and Instruction Level Power Analysis (ILPA). The FLPA is used to model the functional blocks of the soft-core (Arithmetic Logic Unit (ALU), register file, fetch unit). On the other side ILPA is used to model assembler instructions that run in the soft-core. The general power model is build by adding the sub models that relate functional units of the soft-core with instructions. ILPA is used to group instructions in categories, which are affected by similar internal components of the soft-core. The coefficients that are used in the model to relate functional modules with power consumption are obtained through FLPA. Then, these coefficients are further customized depending on the category to which an instruction belongs. The validation of the accuracy of this model is performed using on-chip power measurements. The reported average error is 4.1% with 8.45% as maximum.

An FPGA power aware design flow is presented in [12]. The design flow focuses on high-level optimizations and incorporates a power model for prediction. The design flow performs the optimizations after HW/SW partitioning takes place, in order to optimize power consumption of hardware modules to be implemented in the FPGA. The result of the optimization using this model is an optimized HDL file implementing the required hardware modules. The authors enclose in libraries the set of power models proposed. These libraries are used in the design flow to predict power consumption and build a hardware module that meets the requirements of power consumption. Two types of libraries are identified in the design flow, one for IP-Cores (Intellectual Property Cores) and one for operators models. Depending on the availability of IP-cores in the FPGA, the design flow considers the selection of IP-cores or creates a custom computing machine

that implements the required hardware module. In this context, the models are used to select the most optimal components in terms of power. Aside from the design flow, the authors explain that FLPA is used to derive the models for both IP-cores and operators used in HDL (like adder, multiplier, divider, etc). For IP-cores clock frequency and algorithmic related variables are chosen as variables that relate power consumption with the IP-cores, since the latter are considered as black-box. For operator variables the clock frequency, the number of I/O ports, the clock edge number, the activity rate of operators, and the utilization rate of operators are used to relate power consumption with the operators in HDL. The accuracy of the models is validated against the Xilinx XPower tool and on-chip measurements. The error for algorithmic models (IP-cores) is on average 12.4% with a maximum of 34.73%, for architectural models (operators) a average error of 13.7% is reported with a maximum of 31.8%.

A cycle-accurate FPGA energy measurement tool that characterizes energy consumption on FPGA-based designs is presented in [18]. This methodology targets reduction of power consumption in a FPGA-based design taking into account technology parameters of the FPGA fabric, resource utilization, but also considers the interaction between a microprocessor and an FPGA. Three scenarios to reduce power consumption are foreseen by the methodology:

1. The architecture of the design is fixed and changes in partitioning, mapping, and place and routing aim to reduce power.

2. The power reduction is achieved by changing the architecture of the design.

3. The designer reduces power consumption by modifying the interaction of the FPGA with the microprocessor.

In order to allow the reduction of power consumption in the three previous scenarios, the methodology relies on characterization of energy of the FPGA using switched capacitor methods to measure the energy. The measurements are captured by an automatic data acquisition system connected to a PC-based control application. The tool provides cycle-accurate energy consumption. The energy consumption is analyzed and existing power reduction techniques are used to reduce power consumption. In this work, an online measurement tool is proposed, instead of a model, to know the power consumption. However this tool is useful also in system-wide optimizations, since it provides cycle-accurate measurements which can help software designers to use less-consuming low level designs or modify access method to the FPGA-based peripherals. The Root Mean Square (RMS) error of the measurement tool is 4.9% when compared with on-chip measurements performed by high-precision multimeters.

The work presented in this subsection is in general less accurate compared to that presented in subsection 2.1.2. However the time spent in design and optimizations is reduced because detailed information of the design implementation is not required for power prediction. The only exception is [18], which proposed a tool that characterizes energy consumption based on on-the-fly measurements of a design.

Nonetheless, any of the presented papers in this subsection is suitable for our purposes because the input of the models it is still HDL which is not available before partitioning process. And although nowadays exist C-to-VHDL translator tools, for a design space

exploration during partition process, the time spend in the translation has a high impact in the overall partition process.

2.1.4 Summary of Power Models for FPGAs

Table 2.1 summarizes the literature review presented in section 2.1. A description of each of the columns is given as follows:

- The **Category** is based on the categorization defined for this section.
- The **Target Optimization Level** shows the abstraction level where the model(or methodology) is used to optimize power consumption.
- The **Prediction Input** shows the required input data which is required to predict power consumption.
- The **Validation Method** presents the methodology used to validate the accuracy of each work.
- The **Avg Error** & **Max Error** shows the reported average and maximum error.
- The **Optimization Methodology** describes if the presented work was included in an automatic design flow or is manual.

It can be seen in the Table that in general the higher the abstraction level, the higher the average error of the prediction methodology is. We can also observe in that in the survey conducted, HDL is the highest abstraction used to predict power and energy of FPGA-based designs. Therefore we proposed a High Level modelling technique which can predict power consumption of FPGA-based design using C-code as input.

2.2 Power estimation in General Purpose Processors (GPP)

The GPP has been the core component of Embedded designs in the past decades. Thus there is a vast amount of research in this field. Moreover, the reduction and estimation of power consumption has been a widely discussed topic in the research. Consequently, we present in this survey the most important methodologies used to estimate the cost (in terms of power) of running an application on such a GPP.

One of the earliest approaches to predict the power consumed by software, running on a GPP, is presented by Tiwari et al. [38]. In this work a methodology to develop and validate an instruction level power model for any processor is presented. The main idea behind the model is to measure the current drawn by a processor as it repeatedly executes certain assembly instructions (or short sequences of instructions). In this way, the authors obtain a power cost of a program for that processor.

The methodology assigns to each instruction of the processor's Instruction Set Architecture (ISA), a fixed energy cost. The energy cost is a cycle-accurate average value, and it's measured when the instruction is isolated from external effects like circuit state,

Reference	Category	Target Optimization Level	Prediction Input	Validation Method	Avg Error %	Max Error %	Optimization Methodology
[31]	FPGA architecture	Architectural	Signal statistics Circuit description	HSPICE	N/A	N/ A	Automatic (VPR tool)
[11]	FPGA architecture	Architectural	Resource utilization Signal statistics	on-chip measurements	18	27	Manual
[16]	FPGA architecture	Architectural	Circuit description file	N/A	N/A	N/A	Automatic (VPR tool)
[34]	Low-level	RTL	Signal statistics	N/A	3.1	5.0	Manual
[17]	Low-level	RTL	Signal statistics	Xpower	3.14	20.19	Manual
[27]	Low-level	RTL	Resource utilization Signal statistics	on-chip measurements	5	10	Manual
[9]	High-level	System Design	Frequency Area Circuit design(HDL) Vector length	On-chip measurements CAD tools	8	N/A	Manual
[43]	High-level	Application	Circuit design (HDL)	On-chip measurements	4.1 **	8.45	Manual
[12]	High-level	RTL	Circuit design (HDL)	XPower	12.4 IP-cores 13.7 operators	34.7 IP-cores 31.8 operators	Manual
[18]	High-level	System RTL Application	N/A *	On-chip measurement Multimeter	4.9 ***	N/A	Manual

* This is an online measurement tool, not a model to predict either power or energy.
** Error of a hybrid model, which predicts the power consumed by an application running on a soft-processor implemented in a FPGA.
*** This is the RMS error of the measurement tool presented in this work. A model is not presented.

Table 2.1: Comparison of presented previous work related to model-driven power optimization techniques

pipeline stalls, or cache misses. For instructions that take more than one cycle to execute, the instruction cost is multiplied by the number of cycles required by the instruction. In order to characterize the power consumed by the external effects, controlled test cases were designed. The test cases are used to measure the power consumption caused by the external effects. The following external effects were measured:

- **Effect of Circuit State**: executing different instructions in sequence increases the switching activity in the processor. This work assumes that the change of one instruction to another increases the switching activity the most. Therefore, measuring the extra energy consumed by each pair of instructions, allows the inclusion of average energy per pair in the final energy estimation of a program block.
- **Effect of Resource constraints** (stalls): resource sharing in a processor leads to stalls of certain instructions. This problem will lead to an increase of the execution time of an instruction. Therefore, an increase of energy per instruction will occur. In order to account for this problem, this work proposes controlled experiments to determine a base cost of each type of stalls. Then, performing manual code traversal, the different type of stalls and occurrences in a program can be found. Finally, the energy consumption caused by resource constraints can be obtained by adding the energy caused by each stall type. The cost of each type of stalls is determined by multiplying the base cost of a stall, times the number of occurrences of the stall.
- **Effect of cache misses**: since a cache miss leads to an increment of execution

time of an instruction, the same procedure explained for resource constraints is applied for cache misses. The average energy consumed by a cache miss is obtained using controlled experiments. Then, the number of cache misses is determined in a program through code traversal. Finally, the average cost of cache misses and the number of cache misses are multiplied to obtain the total cost of energy caused by cache misses.

This work proposes a manual prediction flow which is defined as follows: the assembler code is split into basic blocks. The base cost of each basic block is determined using the base cost of each instruction. After circuit state and pipeline stalls effects are analyzed, its cost is added to the basic block cost. When a basic block is executed more than once, the number of block executions is determined and the program cost is determined by adding all the basic block costs (considering iterations). Finally, cache miss analysis is conducted and the result is added to compose the final energy estimation. However, no information about accuracy or validation method is presented in this work. This work was designed for small programs, which still can be manually analyzed by a software designer. However, if the complexity or the number of programs to analyze increases, this work becomes infeasible.

Another approach, proposed in [40] uses a cycle-accurate ARM simulator enhanced with power and performance models. The author considers the components of a processor as black boxes, thus the power models of each component were obtained from vendors datasheets. Several components are identified during simulation. These components are: the processor, the L1 cache, the L2 cache, the memory, and the DC-DC converter. Interconnect is modeled as a separate module. Using this organization, a power model is assigned to each component. Finally, the power consumption of the processor is the sum of the power consumed by each of the components in one cycle. In order to obtain the power of each component, the software under evaluation has to be simulated using the ARM simulator. An error of 5% is found when the simulator is compared with a prototype. The same operational frequency was used in both, the simulator and the prototype. This work, although it can be integrated in an automatic design flow, is not suitable for our purposes, because the simulation increases the Design Space Exploration (DSE) time.

In [35], Steinke et al. present an energy model at instruction level targeting a RISC processor with Harvard architecture. The modelling process takes into account bit toggling of internal and external busses, and accesses to off-chip memory to create the model. The goal of the presented work is to use this model to help in the optimization process of software within a compiler. The optimizations are focused on bus coding to reduce power consumption.

The structure of functional units within the processor and off-chip memory is the basis of the model. Since a RISC processor with load/store architecture is used in this work, separate memory for data and instructions are identified. Also a multiplier, a barrel shifter, and an ALU were identified in the processor. Using this structure, the model was built by adding the energy consumed per instruction, as depicted in the formula 2.1:

$$E_{total} = E_{CPU\_instr} + E_{CPU\_data} + E_{mem\_instr} + E_{mem\_data} \tag{2.1}$$

E_{CPU_instr} is the instruction-dependent energy cost inside the CPU. This value is calculated taking into account the dependencies between 0's and 1's of immediate values, register numbers, register values, and instruction addresses.
E_{CPU_data} is the data-dependent energy cost inside the CPU. Its calculation is based on dependencies between 0's and 1's of data addresses, the data itself, and the direction (R/W).
E_{mem_instr} is the instruction-dependent energy cost in the instruction memory. Its value is computed using word width. Also the bit switching of the data and address of the instruction bus is considered. E_{mem_data} is the data-dependent energy cost in the data memory. The word width and the bit switching (of address and data) in the data bus are considered.
After a mathematical model was established, physical measurements in the processor and memory were performed to obtain the parameters which were not defined in the vendor's datasheets. The author assumes that the voltage doesn't change between instructions, so only the current is measured. With the results of the measurements, linear regression was made to find relation between current and the missing parameters. The model shows an error of 1.7% in a sequence of 12 instructions within an endless loop. This work provides a good accuracy, however, the design flow is still manual and the input for the model is assembly code. Both previously mentioned factors increase the time for DSE.

A real-time cycle-accurate energy measurement technique for digital systems is presented in [10]. The technique proposes an instrumentation using switched capacitors to measure the voltage of a processor free from spiky noise. The work presents an in-house measurement tool with real-time acquisition capability. The proposed tool also samples control and address signals to associate each instruction with a value of energy. Using this tool, a multi-dimensional characterization is performed which includes as parameters: the instruction fetch address, the opcode encoding, the operation, the register number, the memory address, the register value, and the immediate operand. Even though, more parameters were found in this work, these parameters where chosen because they can be affected by software designers. This approach gives the real-time capabilities and avoids the deliberate omission of power hungry instructions or sequence of instructions. However, it becomes more complex to implement because of the in-house measurement tool presented.

Abrar et.al. propose in [1], a cycle accurate activity-based energy model for embedded processors. A characterization methodology is presented that doesn't require special hardware. The presented methodology can be used at different levels of abstraction. The paper describes the energy consumed in a cycle with the Formula 2.2.

$$E_i = Base(I_i) + \sum_{s \epsilon S} a^s A_i^s \tag{2.2}$$

Where $Base(I_i)$ is the base cost of an instruction and $\sum_{s \epsilon S} a^s A_i^s$ is the energy consumed because of switching activity on all modeled signals/buses. The key idea is to use a Least Squares method to find the correspondence between energy values measured in a processor and the transitions on signals. An energy analyzer is presented based on the proposed method with an error of 10%. This work is similar to the work presented in

[35].

[30] presents a methodology to characterize power consumption at instruction-level (single instructions and pairs of instructions). The SPARC Leon3 processor is used for power characterization. The methodology uses simulation of back-annotated gate-level netlists and takes into account the effect of switching activity and register relation (which was not validated experimentally). The paper presents two models, one for single instructions and another for pair of instructions to take into consideration inter dependence between instructions:

- **Single Instruction Model.** For each instruction of the ISA, a loop of 100 executions is used to obtain the average energy consumption of each instruction. The body of the loop is composed of five NOP instructions followed by the instruction under analysis and another five NOP instructions. This sequence tries to isolate a single instruction to reduce external effects in the calculation of the base cost of a single instruction.

- **Pairs of Instructions Model**. Uses them same idea as the single instruction model, but the loop body is composed of five NOP instructions, followed by $Inst_1$, then $Inst_2$ and finally another five NOP instructions. The $Inst_1$ & $Inst_2$ represents each possible combination between instructions of the ISA.

The Data switching analysis proposed in this work, was made using only AND instructions. Based on simulation results, it was found that the power consumed when all bits are changing is twice as big as when no bit was switching at all. However, to reduce the simulation time, the switching distribution was found and a representative value was chosen to obtain average instruction-level energy consumption. The model of energy was validated against gate-level simulation with an average error of 3.68%, when no-switching activity was considered. The average error when switching activity was included was 4.14%. The model of pairs of instructions has an error or 6% with no-cache analysis included and 12% with cache analysis. This work suffers from the same drawbacks found in previous works, it takes assembly as input and the design flow is manual, which greatly impacts the duration of DSE.

The work presented by Tan T.K. et.al. in [37], describes a high-level software energy estimation methodology using characterization-based macro-modeling. The proposed modelling methodology works at the functional level of a software program. Two approaches are proposed for macro-modeling of embedded software. Each approach has different efficiency and accuracy characteristics:

- **Complexity-based**. This approach is focused on data-intensive functions. The variables that determine the complexity of an algorithm are used as macro-modeling parameters.

- **Profiling-based**. This approach is focused on control-intensive functions (branches, loops, etc). The internal profiling statistics of a function are used as parameters for macro-modeling.

The main steps of the methodology are:

1. Determine the parameters that characterize energy.

2. Determine typical input data according to function and application area.

3. Obtain the energy consumption of function for every input data at lower level of abstraction.

4. Then, using a relation function and regression analysis, determine the coefficients that relate each of the parameters of step 1 with the energy dissipation.

The methodology was used for SPARC lite and SimpleScalar processors. The model based on complexity-based parameters has an average error of 4.85% for SPARC Lite, and 5.65% for SimpleScalar. Profiling-based model has a max error of 22% for SimpleScalar and 7.4% for SPARC Lite. The work presented in by Tan T.K. has the advantage that uses c-code as input for the model, it also predicts the power consumption of a function. However, the methodology used to determine the representative input data of the functions under characterization is manual, which makes it slow. It also expects that using a big number of input sets, it ensure that the input data is representative, which is not always true. Finally the methodology it's not enclosed in an automated design flow, which reduces the scalability of this approach when a change of hardware is required.

In [25], a black-box modelling approach is proposed to estimate the energy of instructions. A processor is divided into small modules like ALUs, register files, controllers, etc. Then, a power model for each module is assigned and integrated in a profiler that provides information of the use of the modules per instruction. The profiler simulates and profiles the program execution and obtains the signal activity data for the modules in the processor. Using the signal activity and power models of the switched capacitance, the power consumed by each module is calculated. The error of the prediction is 8%. This approach has the disadvantage that the design details of the processor are required, which is not always available.

A model integrated in a simulation framework is used in [33] to estimate the energy dissipation of the PRI900 processor. A speedup of 200 is obtained with a loss of 1.4% of accuracy compared with gate-level simulation. The model is based on Tiwari et al. [38], where the base cost per instruction, circuit state effects, and cache miss effects are used to estimate the power consumption. The work presented uses the DIESEL-verilog gate-level power tool from Phillips for simulation. This work doesn't provide a comparison with others works and it requires simulation of a program to measure the power, which increases DSE time.

Static code simulation is proposed in [2] to predict power and energy. The proposed methodology has three main components:

- **Instruction power profile**. Using the Formula 2.3 characterizes energy dissipation per instruction.

$$E = E_B + E_{OV} + E_{extra} \tag{2.3}$$

  Where E_B is the base cost of an isolated instruction, E_{OV} is the energy consumed because of the switching activity measured between pairs of instruction, and E_{extra}

is the energy consumed because of stalls and cache misses. The profiling methodology measures the current per instruction, the switching activity, the number of stalls, and the cache misses.

- **Static analysis of code**. It uses a Control Flow Graph (CFG) built from the program code. The CFG is used as input to the branch predictor, the loop analyzer and the path generator. The branch predictor assigns probability values to branching events and annotates the edges of the CFG. The loop analyzer identifies loops in the CFG and determines the number of interactions of each loop. The path generator identifies general paths that reach the end of the program and loop paths that reach the end of a loop. The loops paths are used to estimate loop cost in terms of power energy and time.

- **Power estimation**. Finally, the information from the static analysis and the instruction power profiler is used to estimate power and energy values.

An error of less than 20% with respect to real measurements is obtained. This work uses previously presented works and proposes an improvement to determine the dynamic behavior of an application using a profiler. However, the input of the model are still assembly instructions, furthermore, static profiling analysis is required, which increases DSE time.

2.2.1 Summary of Power Models for GPP

Table 2.2 presents a summary of the research presented in section 2.2. A description of each column is given as follows:

- The **Processor** column presents the GPP used to build the model.

- The **Model Input** shows the input required by the model to predict power or energy.

- The **Validation Method** presents the methodology used to validate the accuracy of each work.

- The **Avg Error** & **Max Error** shows the reported average and maximum error.

- The **Design Flow** describes if the presented work was included in an simulation framework or is manually used.

From the survey conducted, we can identify two types of prediction methodologies: the manual prediction and the approach integrated in a simulation framework. Most of the work presented, use assembly instructions as the input for the prediction, only one work uses C code as input. The error in average is less than 10%.

Reference	Processor	Model Input	Validation Method	Avg Error %	Max Error %	Design Flow
[38]	NA	Ass. Inst.	NA	NA	NA	Manual
[40]	ARM	Ass. Inst.	Phys Measure.	5	NA	Simulator
[35]	RISC	Ass. Inst.	Phys Measure.	1.7	NA	Manual/Math
[10]	NA	Ass. Inst.	NA	NA	NA	Real-time measurement tool
[1]	NA	Ass. Inst.	NA	10	NA	Manual/Math
[30]	SPARC Leon3	Ass. Inst. (Single Inst)	Simulation	3.68 - No switching 4.14 - Switching	NA NA	Manual
[30]	SPARC Leon3	Ass. Inst. (Pair of Inst)	Simulation	6 - No cache 12 - Cache	NA NA	Manual
[37]	SPARC Lite SimpleScalar	C code (Complexity)	NA	4.85 5.65	NA NA	Manual
[37]	SPARC Lite SimpleScalar	C code (Profiling)	NA	NA	7.4 22	Manual
[25]	NA	Ass. Inst.	NA	8	NA	Simulation
[33]	PRI900	Ass. Inst.	Gate Level Sim	1.4	NA	Simulation
[2]	NA	Ass. Inst.	Phys Measure.	NA	20	Static code simulation*

*Power estimation is predicted using pre-calculated power profiles and automatic analysis of code using a simulation tool.
Ass. Inst.: Assembly Instructions
Phys Measure.: Physical Measurements

Table 2.2: Presented power and energy power models for GPP

2.3 Research Context and Background

In this section, we present the previous work used as the base for this thesis. This work is carried out in the context of the Delft Workbench project (DWB), therefore, in Section 2.3.1 we present the background of DWB. The DWB project targets reconfigurable embedded processors, specifically the MOLEN polymorphic architecture which is presented in Section 2.3.2. In Section 2.3.3, we present QUAD, a memory access pattern analyzer, which is used as a basis to build a tool for power characterization process in our modelling methodology. In Section 2.3.4, we present QUIPU, a modelling approach used to create the models presented in this thesis.

2.3.1 Delft Workbench

The Delft Workbench (DWB) project is presented in [5]. The authors identify a great potential for reconfigurable computing, but identify two main problems that have to be addressed in order to adopt this technology into large scale:

- A machine organization is required, which can provide a generic way in which components of a GPP and reconfigurable devices can be combined.
- A tool-set is required, which provides a (semi)automated development platform that transforms (existing or new) applications in order to use the reconfigurable computing units.

The first problem is addressed by the Molen Programming Paradigm [39]. The second problem is addressed by the DWB project, which is based on the Molen programming paradigm. In the rest of this section we explain the details of the DWB design flow.

The Molen Programming Paradigm is discussed in Section 2.3.2. The design flow of the DWB is shown in Figure 2.1.

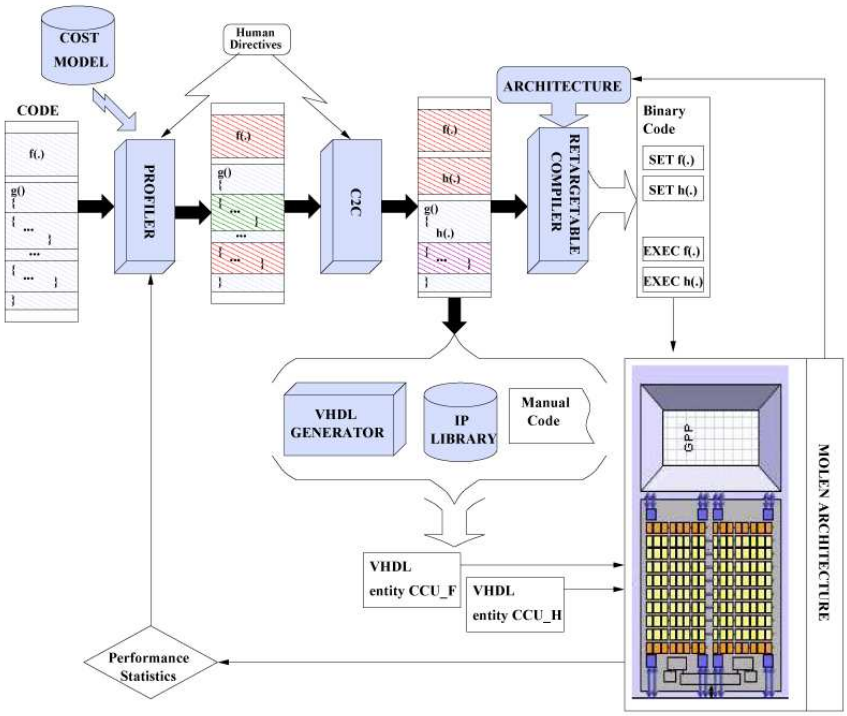


Figure 2.1: The Delft Workbench Design Flow as presented in [5]

The following steps constitute the DWB design flow:

1. **Profiling**. It is defined as the identification of those parts in an application, which can be mapped onto the reconfigurable hardware. The profiling process is combined with an optimization process to ensure that the part of an application that is mapped to the FPGA is within the boundaries defined by the design constraints. In the partitioning process different optimization parameters are considered, which combined with a high number of candidates, that can be mapped in the FPGA, produce a large design space that have to be explored. Therefore, an automatic approach is required in the partitioning process.

2. **Graph Transformation**. The components identified by the profiler are analyzed using *Graph Restructuring* and *Loop Transformation* in order to select and optimize the components which will become new instructions in the instruction set.

3. **Retargetable Compilation**. When the new instructions were selected, the target architecture has to be augmented and the compiler has to exploit these new features. For the Molen programming paradigm, a SET and an EXECUTE instructions are required for each new hardware function. In this context, the compiler

has to schedule the SET instruction in advanced before the EXECUTE instruction is executed, to avoid an overhead caused by the reconfiguration time. The compiler has to decide also, where to place the instruction in the FPGA. To address these spatial-temporal constraints, the DWB introduced advanced instruction scheduling and area allocation algorithms.

4. **VHDL Generation**. This step involves the generation of a hardware specification for the identified new instructions. In the DWB design flow, the hardware is specified using VHDL, a commonly used HLL. If the hardware description of an instruction is not available as IP-core, during this step the HLL specification has to be generated either manually or automatically.

In order to validate both, the Molen programming paradigm and the current available toolset of DWB, a set of algorithms are implemented in the Molen Polymorphic Processor. As a validation example, the symmetric encryption algorithm AES is implemented in a Xilinx Virtex II, which embeds a PowerPC as GPP. A maximum speedup of 750 times was reported for this example.

2.3.2 The MOLEN Polymorphic Processor

A new programming paradigm, a new Instruction Set Architecture, a micro-coded based architecture, and a compiler methodology which together constitute the MOLEN polymorphic processor [39]. MOLEN allows the incorporation an arbitrary number of functions to extend its functionality by exposing the new hardware functionalities to the programmer/designer. The main contributions of MOLEN are:

- For a given ISA a one-time architecture extension is required to address almost an arbitrary number of functionalities. The approach followed in MOLEN *solves the op-code explosion problem* found in similar Heterogeneous Reconfigurable Architectures.
- MOLEN presents a new processor organization along with a new programming paradigm which together *solve the co-existence problem of the reconfigurable co-processor and the general purpose processor*, found in heterogeneous architectures.
- The compiler back-end technology presented allows to target a micro-architecture based on reconfigurable emulation ($\rho\mu$-code), and the compiler implementation allows executing the compiled code.

The Machine Organization. Figure 2.2 depicts the MOLEN machine organization as presented in [39]. The MOLEN organization consists of the following elements:

- A **"Core Processor"** which is a General Purpose Processor (GPP).
- The **Register File** of the GPP.
- A **"Reconfigurable Processor"** (RP).

- A set of exchange registers (**XREG**) used to transfer data between the RP and the GPP.
- An **Arbiter** that issues instruction to the GPP.
- A **Data Fetch** unit, which fetches and stores data.
- A **Memory MUX** unit, which distributes/collects data.

The RP is further divided into a *ρμ-code* unit which controls the execution of single instructions or blocks of instruction (through emulation) using the *custom configured unit* (CCU) which consists of reconfigurable hardware. In these CCUs, the extended functionalities are implemented.

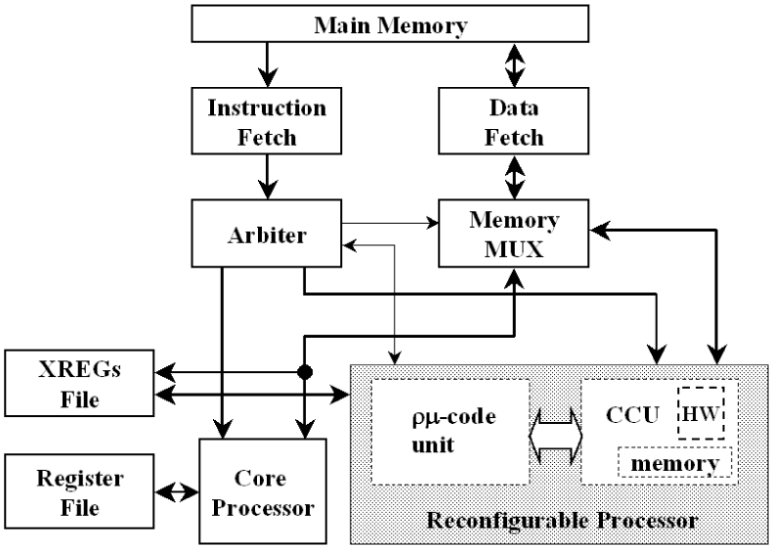


Figure 2.2: MOLEN machine organization as presented in [39]

The execution flow. The GPP executes all the code of an application besides the selected sections which are implemented in the RP to speed up the entire application. The execution of a function, in the RP, is performed in two phases:

1. **SET** phase, where the CCU is configured to execute certain functionality.
2. **EXECUTE** phase, where the actual execution of the functionality takes place.

The division of Set and Execute allows better instruction scheduling to cope with reconfiguration latency.

Polymorphic Instruction Set Architecture (π ISA). In order to expose the functionalities in the RP to the designer, a sequential consistent programming paradigm is proposed. It allows parallel hardware execution and is intended for single program execution. The programming paradigm requires only a one-time architectural extension.

The number of instructions to be added depends on the MOLEN architecture to be implemented.

For a *complete* implementation, the following instructions are added to the ISA to form a polymorphic Instruction Set Architecture (π ISA):

- Six instructions which are required to control the reconfigurable hardware: a partial set, complete set, execute, set prefetch, execute pre-fetch and break.
- Two instruction to transfer data between the register file and the XREGs: movtx and movfx.

For a *basic* implementation of MOLEN only four basic instructions are needed: the complete set, the execute, the movtx, and the movfx instructions.

2.3.3 QUAD - A memory access pattern analyzer

As explained in [28], QUAD (Quantitative Usage Analysis of Data) is a sophisticated memory access tracing toolset that provides a comprehensive quantitative analysis of memory access patterns of an application with the primary goal of detecting actual data dependencies at function level. QUAD provides a thorough analysis of the memory access behavior of an application to improve the development, tuning and optimization processes. QUAD is a profiler that analyzes the behavior of an application at run-time, it detects actual data dependencies which arise when a function consumes data that is produced earlier by another function. The actual data dependencies traced by QUAD core are based on the journey of bytes through the memory addresses. As main features of QUAD we can distinguish:

- Quad provides *actual* data dependencies at function level, which involves higher accuracy when it is compared with similar tools that provide conventional data dependencies.
- QUAD does not require any modification of the binaries, and it does not have compiler dependences other than debug information.

QUAD is a Dynamic Binary Analysis (DBA) toolset implemented using the Pin [19] framework. Pin allows a transparent instrumentation, which does not have any dependence on the compiler or the source code language, it only requires the application to be compiled in a common binary format.

In the Figure 2.3 we can observe the architectural overview of QUAD, including the components of Pin. Two main components can be identified:

1. **QUAD**. It contains the instrumentation and analysis routines and it is linked with Pin via the Instrumentation API (Application Programming Interface). The main module is the Memory Access Tracing (MAT) module, which is responsible of tracing and maintaining the memory accesses information.
2. **Pin**. Pin is the engine that instruments the application. It is composed of a Virtual Machine (VM), a code cache and the Instrumentation API.

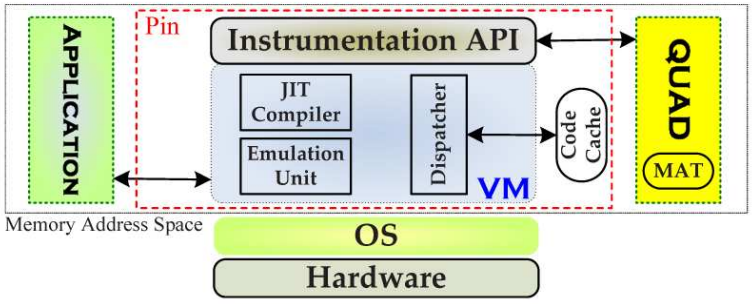


Figure 2.3: Architectural overview of QUAD as presented in [28]

The MAT module is based on a data structure called *trie*. The *trie* is composed of *trie* nodes and corresponding data blocks. The *trie* nodes are implemented as an array of 16 pointers which are used to trace the address of memory accesses. The data blocks store the data which resides in memory addresses. The address of a memory access is virtually stored and traced in a *trie*, using an arrangement of 8 levels of trie nodes. Each level in the trie hierarchy corresponds to a hexadecimal digit in a 32 bits address. The trie data structure is implemented in a dynamic fashion, which means that the trie nodes and the data blocks are allocated on demand, resulting in memory overhead reduction.

QUAD was developed in the context of the DWB project. Therefore, we present QUAD within the Profiling Framework of DWB in Figure 2.4.

2.3.4 High Level Quantitative Hardware Prediction

The work presented by R. Meeuws et.al. in [24] and [22], is used as a base for the research conducted in this thesis.

In [24], a high level model for hardware predictions that helps in the partitioning process of HW/SW co-design is presented. The predictions made with this modelling methodology are used to prune the design space to find an optimal solution for different hardware characteristics. The model is based on software complexity metrics obtained from C-code. The software complexity metrics are used to characterize the hardware metrics that need to be predicted. The characterization methodology is made using the DWARV C-to-VHDL compiler [42] to automatically obtain synthesizable VHDL code from which hardware characteristics are obtained using the Xilinx design tools. Linear regression techniques are used to find the relationship between the obtained hardware characteristics and the software metrics. A mean error of 68% is reported for flip-flop prediction when this approach is validated against the results provided by Xilinx design tools after the design synthesis.

In [22], R. Meeuws et.al. present *a high level quantitative prediction modelling approach that accurately models the relation between hardware and software metrics, based on several statistical techniques*. Compared to [24], in [22] the authors propose beneficial enhancements in the statistical techniques used for modelling. The contributions of this work were:

- It demonstrates that the proposed approach can generate comparable and appro-

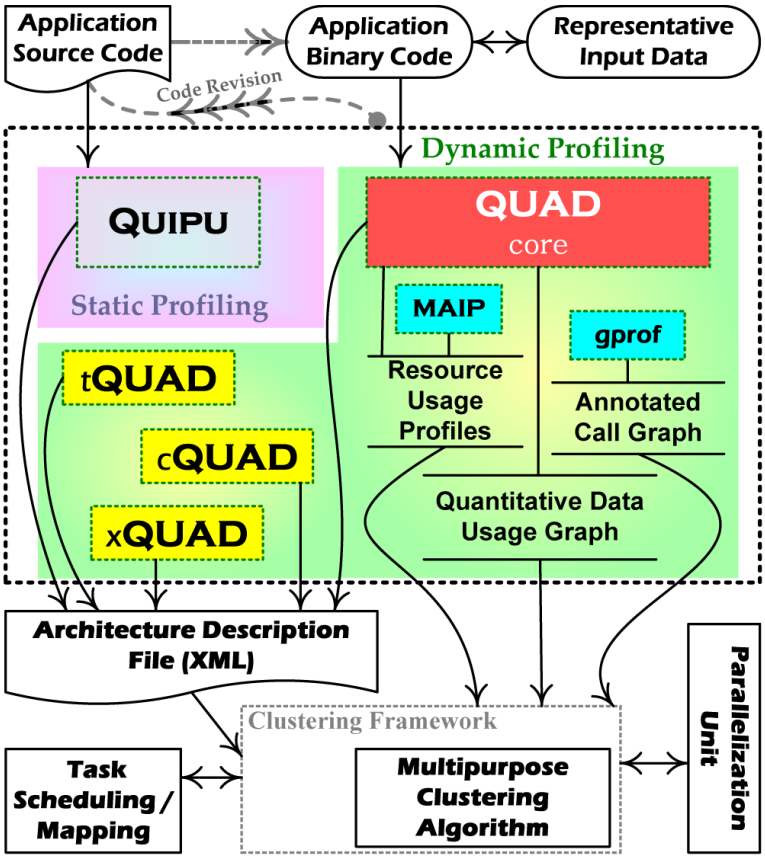


Figure 2.4: QUAD within the Profiling Framework of DWB

priate prediction models for two independent tool-chains and platforms. One tool-chain is composed of the DWARV C-to-VHDL compiler [42] used with the Xilinx ISE Synthetizer for the Virtex 4 platform. The other tool-chain is composed of the Altera Stratix IV FPGA and the C-toVerilog compiler from the Haifa University [3].

- It uses different statistical techniques in the modelling process to reduce the prediction error of the models. The error is reduced, ranging from 15% to 34%. The techniques used are Artificial Neural Networks (ANN), model selection, logistic regression, and data transformations.

- It provides a detailed prediction of hardware parameters of an FPGA, such as clock wires, logic wires, and power wires.

- It improves the relevance of the produced models by using a set of 181 kernels, opposed to similar works which use at most tens of kernels.

A detailed explanation of Quipu modelling approach and the enhancements proposed in this work are presented as follows:

Quipu models and criteria. In [24] and [23], Software Complexity Metrics (SCM) were introduced as indicators of specific characteristics of a software code. 58 SCMs were used to characterize hardware metrics of an FPGA. Using these SCMs, a model was selected, which approximately relates the SCMs and the hardware metrics. The model is presented in the Equation 2.4.

$$\hat{y}_{HW} = \hat{g}(\tilde{X}_{SCM}) + \hat{\epsilon} \tag{2.4}$$

The approximation $\hat{g}(\cdot)$ can be, for example, an ad-hoc model, a Linear Regression Model (LRM), a Generalized Linear regression Model (GLM), or an ANN. The hardware metric to be predicted is $\hat{y}_{HW}$. The vector containing the SCMs is $\tilde{X}_{SCM}$.

Quipu Tools and Kernel Library. Quipu consists of a set of tools and a Kernel Library, which are presented in the Figure 2.5. Quipu extracts SCMs and hardware performance metrics from a kernel library. In this work the library contains 181 kernels from a wide variety of domains. The SCMs can be extracted using *Quipu Metrication tool*. The hardware performance metrics can be extracted using *Quipu Hardware Measurement tool*. The SCMs and the hardware measurements compose the HW data. The HW data was analyzed with a set of modelling scripts that automatically evaluate different statistical modelling techniques and generate a model.

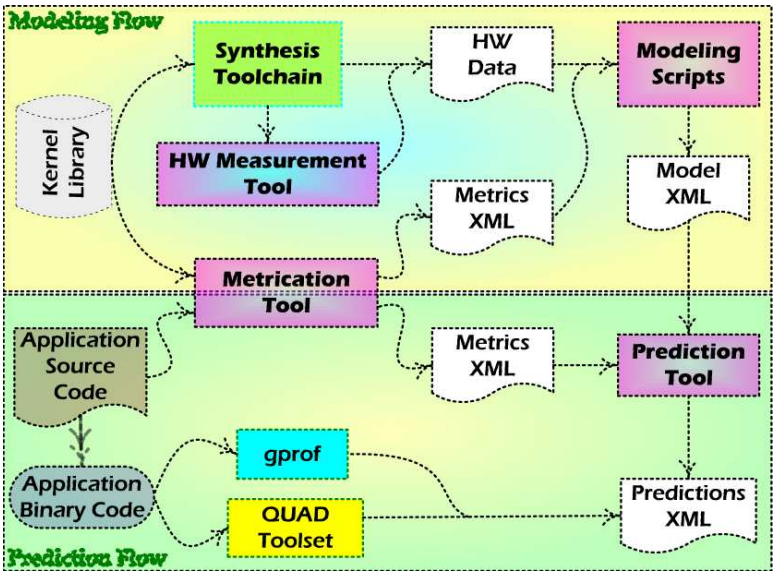


Figure 2.5: Block diagram of the Quipu modelling approach as presented in [22].

Quipu regression techniques. In [24] and [23], LRM, GLM, and Partial Least Squares Regression (PLSR) were introduced as the techniques used for Quipu modelling. The following statistical techniques are introduced and employed in Quipu:

1. *Stepwise Model Selection (SMS)*. Some SCMs used in this work can measure more or less the same software characteristic of a code, this is called collinearity between SCMs. The collinearity is a problem during regression analysis because certain software metrics (or predictors) are overrepresented. Therefore, in order to reduce

the chance of collinearity the number of SCMs (predictors) can be reduced. For this purpose SMS is used. The SMS is an iterative process where different predictors are added and removed step-by-step. Then, the significance of each predictor is measured and the predictors that do not increase the error are removed. This process continues until no single predictor can be added or removed to improve the model. In this work, the SMS process was used only for GLM models.

2. *Data Transformations.* Many of the metrics used in this work did not show a linear relation with the hardware metrics to be predicted. Therefore, the author used the Box-Cox power transform [7], a known data transformation technique, which reduces the variance of the dataset and makes the sample distribution more similar to the normal distribution. Given the data set used in this work, the author used for all the metrics a log transformation, which is one of the transformations in the Box-Cox power transformation.

3. *Artificial Neural Networks (ANN).* Despite the data transformations discussed, the non-linear problem was still present to some extent. Hence, ANN was included in Quipu. The author used a ANN training package called *nnet* from the R statistical computing environment.

4. *Logistic Regression and Count Regression.* Apart from collinearity and non-linear behavior of the data set, the author identified many non-negatives values in the data set. For this reason, it was possible and beneficial to use GLM techniques instead of regular linear regression techniques. With GLM the error can be modelled more accurately using Poisson or Negative Binomial distributions. However, GLM requires that the dataset does not contain many zeroes, hence, Logistic Regression was used to determine if GLM could be applied.

To evaluate the predictive quality of the statistical models generated with Quipu a technique known as cross-validations was used. In specific, a method called *K-fold cross-validation* was chosen by the author using 10-Fold cross-validation (K=10). In the K-fold cross-validation the most common error summary is the Relative Rooted Mean Square Error of prediction (RMSEp), therefore, Quipu reports RMSEp for the generated models. Also Quipu reports the cross-validated coefficient of determination (R^2) which shows how much of the variance of the original dataset is explained by the model and does not contribute to the error.

The Quipu modelling methodology was implemented in two scenarios:

1. With the DWARV C-to-VHDL compiler and Xilinx ISE 11.5, targeting Xilinx Virtex 4 LX200 FPGA. For this scenario the RMSEp ranges from 3% up to 34% for different hardware metrics.

2. With the C-to-Verilog compiler developed at the University of Haifa [4] along with Quartus II 9.0 Build 235, targeting the Altera Stratix IV EP4SE530 FPGA. The RMPSEp ranges from 13% up to 82% for different hardware metrics.

2.4 Conclusion

We presented in this chapter, a review of the previous research focused on power and energy modelling. This thesis aims to improve the HW/SW co-design flow of the DWB. We presented the previous works for modelling power and energy consumption of two components found in heterogeneous architectures, a GPP and an FPGA. The survey presented can be summarized as follow:

- **General Purpose Processor (GPP)**. This component has been the subject of a thorough research with respect to power and energy predictions at all levels of abstraction. In the higher abstraction levels (e.g. C code) there are already works which can predict the power consumption of one single instruction and forese the overall power consumption of a program. However, the methodologies proposed are completely manual, which increases the modelling time when a change of underlying hardware is necessary.

- **FPGA**. The problem to predict power consumption has been addressed from lower abstraction levels (RTL) up to SDL. However, there is no current work that predicts power consumption using a HLL (e.g. C code) level. Furthermore the current methodologies are manually applied and they are not fully integrated in CAD tools.

According to the literature research conducted we conclude that using models for the prediction of hardware metrics, such as power or energy consumption, involves a tradeoff between accuracy and prediction time. This tradeoff can be explained with Figure 2.6.

In **lower abstraction** levels the detailed information of the hardware implementation of a design is higher. Therefore, a model using this detailed information can provide a higher prediction accuracy. However, the prediction of power and energy consumption requires dynamic information of a design, such as switching activity or effective capacitance. This dynamic information can be calculated using mathematical models, but only if the detailed information of the hardware implementation is available. Moreover, for small designs the mathematical models are still feasible to use, but for complex designs the common approaches to obtain this information are simulation and complex measurements on the chip. Therefore, a prediction methodology that requires simulation or physical measurements to obtain power or energy requires a high prediction time.

On the other hand, a **higher abstraction** model abstracts many details away of the underlying hardware. The abstraction of details allows faster predictions. Nevertheless, the accuracy of the prediction is reduced because the model is missing details of the hardware implementation.

This thesis aims to model power and energy consumption at SDL, therefore, our challenge is to the reduce the prediction time, meanwhile keeping enough accuracy to perform the design space exploration in the HW/SW partitiong process. In order to find an optimal solution for the previously mentioned modelling trade-off, we believe it is important to solve the following issues in this thesis:

- Since the power and energy consumption is strongly determined by the characteristics of the hardware component to be modelled, an automatic high level modelling

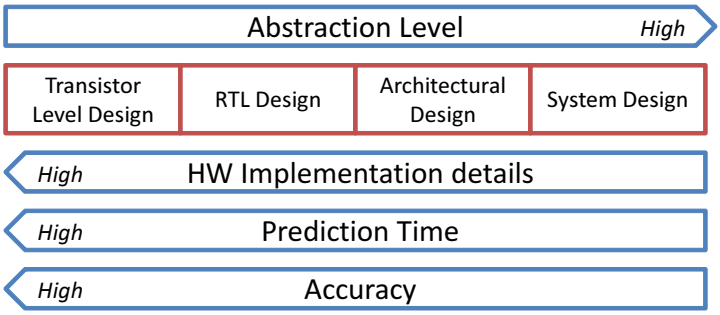


Figure 2.6: Tradeoff in prediction models of hardware metrics.

methodology is required in order to investigate different hardware metrics that affect power consumption. This will help to reduce the modelling effort when different hardware characteristics should be analyzed.

- A high level modelling methodology should not require low level details of the hardware component to be modelled. If the HW component is considered as a black-box during the modelling process, the methodology would be more flexible and scalable. Hence, it would be possible to use it for different hardware components.
- For our purpose, we require a model which takes as input a HLL (C code) and predicts power and energy consumption of a hardware component. The most important requirement of this model is the prediction time, because it should be small to allow a broad design space exploration during HW/SW partitioning. The model should also be able to provide predictions that allow qualitative comparison during HW/SW partitioning. Even though, the absolute accuracy of the model is not required to be as high as the models that require low level details of the hardware.

Experimental Methodology and Setup

3

The experimental setup described in this chapter is built on the conclusions derived in Chapter 2. These conclusions are summarized here for your convenience, in the form of research questions:

- *Would an automatic high level modelling methodology help to investigate the different parameters that affect power consumption?*
- *Are the low level details of a HW processing unit necessary during the modelling process? Does considering the HW processing unit as a black-box provides more flexibility and scalability to the methodology?*
- *Does using a HLL (C code) as input for a prediction model reduces the prediction time of the model? Do the predictions obtained with this model provide qualitative data useful for the partitioning process in the DWB design flow?*

Based on these questions, in this Chapter we propose a modelling methodology that can be used to build prediction models of the power and the energy consumed in processing units, such as a GPP or an FPGA. In Section 3.1, we explain the rationale and description of the proposed modelling methodology. In section 3.1.1, we describe the first step of the modelling methodology: the kernel isolation process. Finally, in Section 3.1.2, we describe the evaluation criteria to select the metrics for power and energy characterization.

Using the modelling methodology proposed, we design a set of experiments that are described in Section 3.2. These experiments are categorized as follows:

A. **Physical Measurements**. In this category we measure the power consumed by a kernel using physical measurements in a chip. The following experiment is the only one designed for this category:

   1. **Experiment PowerPC.Virtex5.GPP.A.1**.

B. **Simulation framework**. In this category, we use a simulation framework to measure the power consumed by a kernel during its execution in the processing unit. The following two experiments were designed for this category:

   1 **Experiment StrongARM.SimPanalyzer.GPP.B.1**.

   2 **Experiment VIRTEX5.Xpwr.FPGA.B.2**.

In order to assist in the identification of the experiments, and provide a proper structure in this report, we define the following naming convention for the experiments: $Modelling_{HW}.Measurement_{HW}.UUTtype.Category.Experiment\#$. Where,

the $Modelling_{HW}$ is the hardware which is being modelled, the $Measurement_{HW}$ is the chip being measured or the simulation framework being used, the $UUTtype$ defines in general for which UUT this experiment is made, the $Category$ is the category of the test, and the $Experiment\#$ is the number of experiment in that specific category.

3.1 Modelling methodology: rationale and description

The final goal of the power models, built with our modelling methodology, is to improve the partitioning process in the HW/SW co-design of embedded systems. As we explained in Chapter 1, *HW/SW partitioning is an activity where the tasks identified in early design stages are mapped to either hardware or software.* In the context of Delft Workbench, the tasks are C-coded functions (throughout this thesis we refer to these functions as kernels). Thus, *the models built with the proposed modelling methodology will predict the power consumption of kernels, defined in an application source code.*

In order to provide a fast and accurate estimation of the power and energy consumed by a function, we should *characterize the power, consumed by the function, using a set of high-level metrics that describe the function and the underlying hardware.* In the scope of this thesis, we used the Software Complexity Metrics (SCM) presented in [24] and we envision an automated approach that allows the configuration, or even the change, of the underlying hardware. This work uses, as base, the QUIPU modelling approach presented in [22], with an already selected set of software metrics describing a C-coded function.

Using these assumptions we design the modelling methodology presented in Figure 3.1.

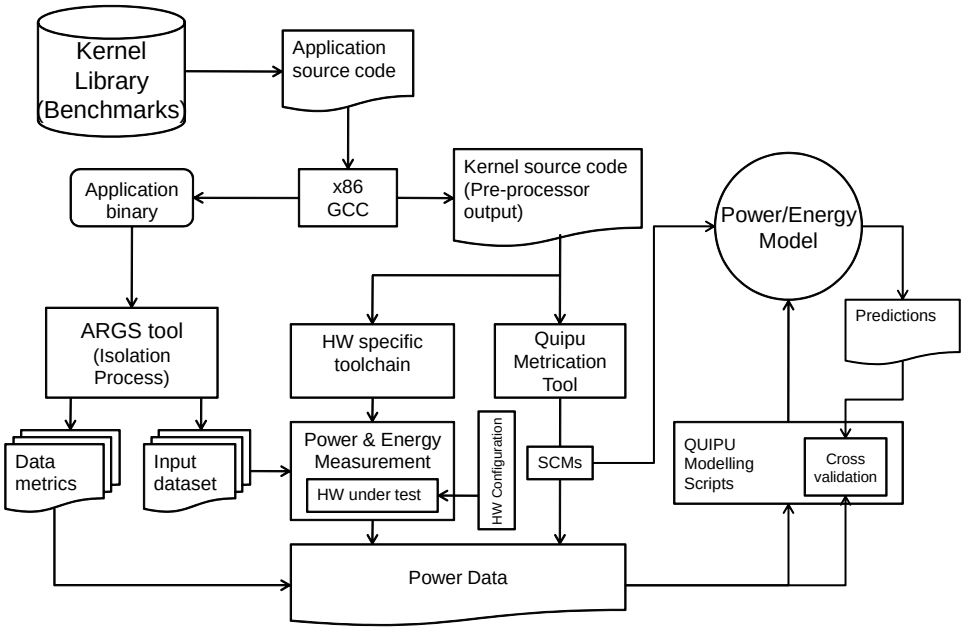


Figure 3.1: Modelling methodology diagram

The methodology uses a Kernel library which contains a set of common applications and benchmarks. The applications are collected from different domains, such as, Multimedia, Cryptography, Error Correction, Physics, and Mathematics. These libraries were selected because they are commonly used in high performance computing, which is the target of the DWB project.

For each application in the Kernel library one (or more) kernel(s) were selected for characterization. The selected kernels have the following properties:

- **Candidacy for hardware implementation**. It should be possible to get a synthesizable hardware component for the FPGA using a C-to-VHDL translator. Therefore, the kernel should not have function calls, because they cannot be mapped to hardware by the DWARV C-to-VHDL compiler [42]. Also the functions do not use globals variables, because in the Molen organization the value of a global variables cannot be passed to a function, unless is passed as a argument of the function.

- **No loops**. In the scope of this thesis, we only select kernels that do not have loops, because the number of times the body of a loop is executed influences the energy consumption. Analysis of kernels with loops is left as future research of this thesis.

The Table 6.2 in the appendix 6.4, provides a summary of the *kernel library*.
We can describe the proposed overall modelling methodology with the following steps:

1. **Kernel Isolation Process.** For each kernel execution, we obtain the data passed to the kernel (through the arguments). This step is carried out using a tool developed in this work, called ARGS. In Section 3.1.1, we provide further details of this step.

2. **Metrics Extraction.** Extract the metrics that will be used for characterization. We define two types of metrics: the static metrics (SCMs), and the dynamic metrics (such as the number of cycles required to execute the kernel, or a metric that describes the data passed to a kernel). The static metrics are obtained using the qpm-metricator tool [22]. The dynamic metrics are extracted using the ARGS tool and a set of bash scripts. Section 3.1.2 presents a detailed description of the metrics used in this work, and the details of the extraction process.

3. **Power Measurement.** Obtain the power consumed by the kernel using a controlled experiment. This experiment uses the datasets obtained in step 1. The datasets are used as input for the kernel during the measurement of power and energy. It is important to mention that in this work two type of experiments were designed, one using physical measurements, and another using simulation. These experiments are categorized as follows:

   (a) **Physical Measurements**. In this category we measure the power consumed by a kernel using physical measurements in a chip. The characteristics of this experimental setup are presented in Section 3.2.1. One experiment was designed in this category, and is described as follows:

i. **Experiment PowerPC.Virtex5.GPP.A.1**. In this experiment we measured the power consumed in a PowerPC processor, embedded in a Virtex 5 chip. The details of this experiment are presented in Section 3.2.1.2

(b) **Simulation framework**. In this category, we use a simulation framework to measure the power consumed by a kernel during its execution in the processing unit. The details of this experimental setup are presented in Section 3.2.2. We designed two experiments that are summarized as follows:

i. **Experiment StrongARM.SimPanalyzer.GPP.B.1**. This experiment uses the Sim-Panalyzer simulator, configured as a StrongARM processor, to obtain the power consumption by a kernel running in the StrongARM processor. The StrongARM is configured with a clock of 200MHz and a voltage source of 1.8v. The complete configuration file is presented in Section 6.1. The details of this experiment are presented in Section 3.2.2.3.

ii. **Experiment VIRTEX5.Xpwr.FPGA.B.2**. This experiment uses the *xpwr* tool of Xilinx to estimate the power consumption of a kernel implemented in a Virtex 5 XC5VFX130TFF1738-2 FPGA (with the MOLEN machine organization). The details of this experiment are presented in Section 3.2.2.2.

4. **Power Data Matrix consolidation.** Create a power data matrix with the measured values and the extracted metrics.

5. **Model Building.** The Quipu modelling methodology is used to obtain a power and energy model in this step. The power data matrix is used as input for Quipu. In section 2.3.4, we presented the details of the Quipu modelling approach.

3.1.1 Kernel Isolation Process

Initially, in this section, we present the reasoning behind the first step in our modelling methodology, *the Isolation Process*. Subsequently, we describe the implementation details.

In order to reduce the complexity of measuring the power and energy consumption of a kernel during its execution in a processing unit (GPP or FPGA), we can execute the kernel isolated from its application. However, if we only execute the kernel, we need to provide input data through the arguments as well. The data passed through the arguments of a function (kernel) determines its dynamic behavior and, thus, the power that will be consumed during the execution of the function. For instance, if we pass randomly generated data to the function we might replicate the same dynamic behavior of the function over and over again and miss important execution paths. As a result, it is important to provide representative data to the function in order to exercise the most typical execution paths (power-wise) within the function. Moreover, since the type and values of the data determine the dynamic behavior of the function, it is also important to select valid data that ensures a correct execution of the function.

With the *Kernel Isolation Process* we mean the selection of representative and valid data of any function, which allows its isolation from an application. This process could be manually performed through code traversal. However, a manual approach would increase the complexity and time of the isolation process. Therefore, we assume that it is important to use an automatic approach to select the input data of any function in order to reduce the time of the kernel isolation process. In addition, the automatic approach should be able to select representative and valid input data.

For this purpose, we designed the ARGS tool that automatically performs the Kernel Isolation Process. The ARGS tool executes the entire application, and for each kernel execution, the input data (passed through the arguments) is recorded. With this approach we ensure that valid and representative data is selected. The ARGS tool uses Pin (a toolkit for dynamic instrumentation of a program) to trace and record the input data passed to the kernel. In the following subsection, we describe the implementation details of ARGS.

3.1.1.1 The ARGS tool

The ARGS tool was designed for two purposes:

- **Kernel isolation**. The ARGS tool isolates the kernel from its application by extracting the input data passed to a function. For each execution of the kernel, ARGS traces and stores the data present in the memory of the arguments before it is modified by the function. We call this data the *input dataset*.
- **Validation of the kernel**. After the kernel is executed, the function could have modified the memory regions of the arguments or the return value. Therefore, ARGS traces and stores the data present in the memory arguments after it is modified by the function. We call this data the *output dataset*. This dataset is used for validation purposes.

We use the Pin toolkit to instrument the application in order to extract the input and output data of a kernel. However, Pin does not provide a function that returns all the data passed to a function or modified by a function. Therefore, we use the Pin API to trace all the memory accesses occurred during the kernel execution, in order to associate the corresponding accesses with each of the arguments in the function. As a result, we developed the ARGS tool, a customized tool that uses the Pin API to trace memory accesses, which are used to obtain the *input* and the *output* data of a function.

Since the ARGS tool traces memory accesses in an x86 processor, we consider it is important to present an introduction of the memory-segments layout in an x86 architecture used by C compilers:

Figure 3.2 shows the memory-segments layout commonly used by C compilers in an x86 architecture with Linux, as presented in [36]. We are mainly interested in two areas, **Heap** and **Stack**. Heap holds the dynamically allocated variables in a program and stack holds the *Activation Records* of the functions. These segments contain the data passed to a function through the arguments, thus we need to trace the memory accesses made in these regions. As shown in Figure 3.2, stack grows downwards (from high to low memory addresses), and heap grows upwards (from low to high memory addresses). The

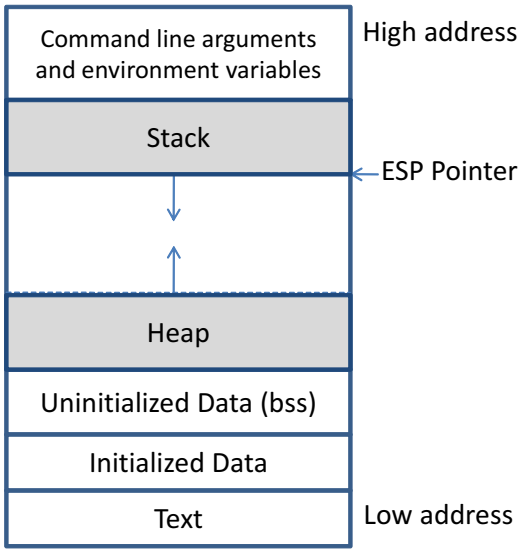


Figure 3.2: Common memory-segments layout used by C compilers in a x86 architecture

Extended Stack Pointer (ESP) defines the boundary between these segments. The stack region expands each time a function call is made in the program. The actual process to call a function is determined by the compiler. As explained in [14], for each function call found in the source code of a program, the compiler generates assembly code that performs the following actions:

1. Provide a new environment for the called function with some temporary memory to store local variables.

2. Pass the parameters of the called function to the new environment.

3. Transfer the control flow to the called function.

4. Return information and control flow from the called function to the calling function (in a successful and normal execution).

The new environment is a data structure called *activation record*, which is stored in the stack in a Last In First Out (LIFO) fashion. The *activation record* contains the information required to associate a function call in a program. The typical structure of an *activation record* is shown in Figure 3.3, based on the structure presented in [14]. One should note that the details of the construction of the *activation record* is out of the scope of this work.

When the *activation record* is made, Pin is able to deliver the memory addresses of the Extended Base Pointer (EBP) and the ESP to ARGS. These pointers are used to determine the accessed memory region when a memory *Read* or *Write* is performed. Apart from EBP and ESP, it is necessary to know the Effective Address (EA) of the arguments (or parameters) passed to the functions. This information is important because we want

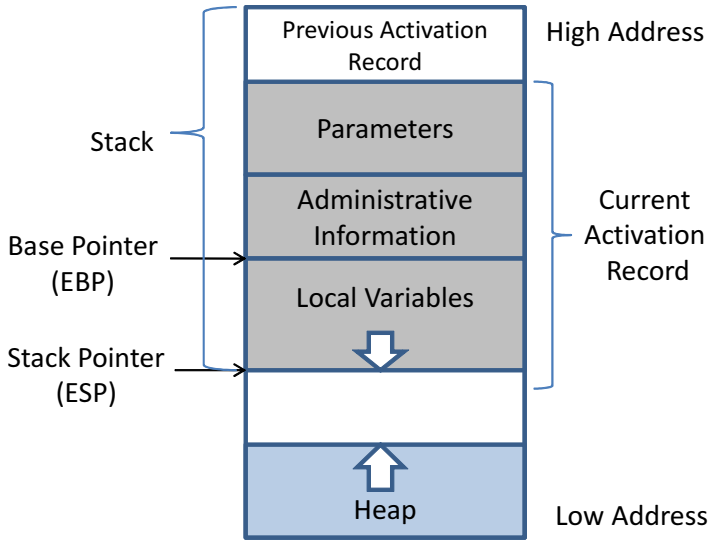


Figure 3.3: Typical Activation Record structure

to associate a memory read or write to the appropriate function argument. The EA of the arguments is provided by Pin.

Following this brief overview, we now explain how the data passed to the function, through the arguments, is traced in ARGS. The general steps for this procedure are as follows:

1. Get and store the EA the address pointed by the arguments passed by reference.

2. Trace all the memory read and write operations while the function is running.

3. Filter the memory accesses, during the execution of the function, that read or write to the memory regions of each argument, using the EA extracted in step 1, ESP and EBP.

4. While the function is running, we store, temporarily in RAM, the input and output data, and the addresses of the filtered accesses.

5. After the execution of the function and before its return, we associate the filtered memory accesses to each memory region of the arguments.

6. Before the function returns, we store in one file the input data of the function. In another file, we store the output data of the function.

Using the general steps outlined above, we design the software architecture of ARGS tool, which is presented in Figure 3.4. In the following subsections, we describe the implementation details of each module in ARGS.

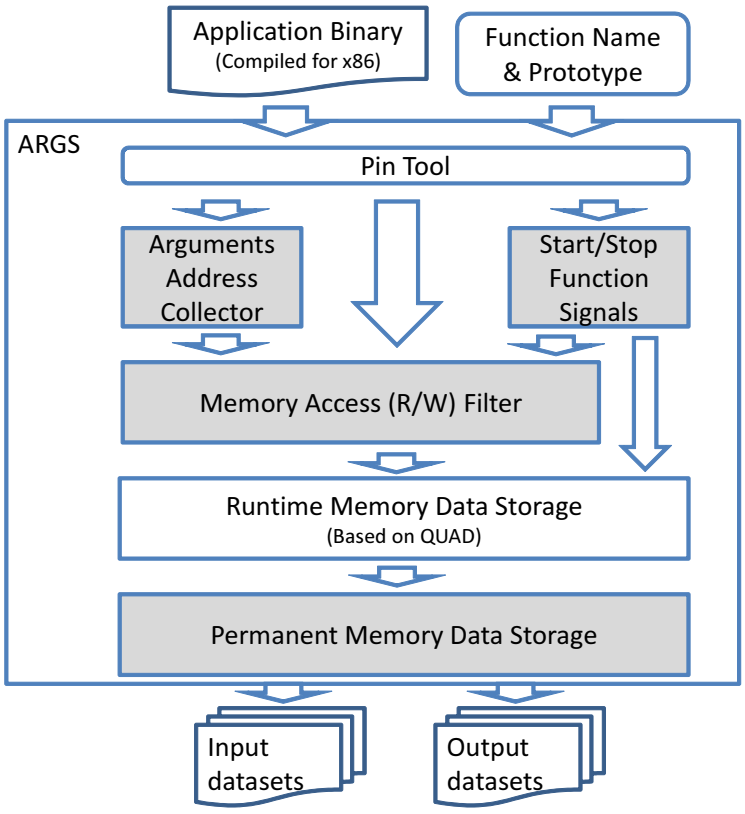


Figure 3.4: ARGS tool Software Architecture

The Pin Toolkit . ARGS uses Pin, a dynamic instrumentation toolkit presented in [19], to extract runtime information of an application. The data provided by Pin is used for different purposes in the modules of the ARGS tool.

ARGS receives as input a binary of an application compiled for x86 architecture. ARGS also receives the name and the prototype of the function to be isolated. This information is used by Pin to provide runtime data to the modules of ARGS. The following data is provided by Pin during the execution of an application:

- **Runtime Addresses of the arguments.** The EA of each argument in the function are passed to the Arguments Address Collector module.
- **Memory Access Data.** During the function execution, all the memory accesses (R/W) are traced by Pin, and the data and the addresses are passed to the Memory Access Filter module.
- **EBP & ESP.** The EBP and ESP of the current activation record are passed to the Memory Access Filter module each time a memory access is detected.
- **Start/Stop signals of the function.** Pin provides a start and stop signal to the Memory Access Filter and the Runtime Memory Data Storage modules.

The implementation details to gather the information outlined above are presented as follows. The Pin toolkit works as a Just In Time (JIT) compiler. Because it allows the dynamic inclusion of arbitrary code (C or C++) before or after executing any machine instruction. In addition, Pin also allows the inclusion of code using higher abstractions, such as, a binary image, or a function.

At any abstraction level, it is necessary to define in Pin a mechanism which decides the code to insert and the exact place to do it , this code is grouped in *instrumentation* functions. The functions which analyze and gather any dynamic information of an application are called *analysis* functions.

In the ARGS tool, we insert an *instrumentation* and an *analysis* function for the following abstraction levels:

- **Binary image**. It is necessary to insert an instrumentation function before loading a binary image, because we need to search in the symbol list (only available when loading the image), the name of the kernel to be analyzed. If the kernel symbol is found, we insert an analysis function before the kernel is executed; this function stores the EA of any number of arguments in the kernel and signals the start of the kernel execution. We also insert an analysis function after the kernel is executed to signal the end of the kernel execution.

- **Function**. Before and after any function is executed we insert an analysis function to check if the kernel is running. This function was added to allow the scalability of ARGS, because it allows executing any other function inside the kernel.

- **Instruction**. We insert an instrumentation function that inserts (different) analysis functions before and after a memory read or write instruction is executed. Before a memory R/W instruction is executed, the inserted analysis function obtains the data present in memory before the instruction is executed. After the memory R/W instruction is executed, the inserted analysis function obtains the data present in memory after the instruction is executed. Both analysis functions obtain the ESP, the EBP, and a flag which shows if the memory operation is a read, or a write.

Arguments Address Collector . This module stores the EA of the arguments and passes this data to the Memory Access Filter.

Memory Access Filter (MAF): . This module acts as a runtime filter of the memory accesses traced by Pin during the execution of an application. The memory accesses are triggered by read or write instructions, and for each memory access, this module receives the data, the address, along with the ESP and EBP pointers. The following rules define which memory accesses are passed to the Runtime Memory Data Storage (RMDS) module:

- **No pointers.** If the prototype of the function does not contain pointers, there is no need to trace any memory access because Pin can directly provide the data of the arguments passed by value.

- **Kernel is running**. Only the memory accesses occurring during the execution of the kernel are passed to the RMDS module.
- **Access below the Lowest Argument EA (L.Arg.EA).** Any access below the lowest EA of the arguments is ignored. Because, we assume that the address pointed by the EA of an argument, is the lowest address a function will access. We also assumed that the functions to be profiled do not use globals variables, because the Molen organization do not pass global variables to a function, unless they are passed as arguments. As a result, the globals region, that is below the heap, will not be accessed at all.
- **Heap Access.** All accesses to the heap are stored, except if the base address of the access is the EBP. This restriction is defined because when the called function starts execution, writes the values of the registers, which are going to be restored when the function returns, in the current heap region. This operation is, in principle, pushing the register in the stack, however what actually happens is that the register data is stored in the current heap and then the ESP is modified to point to the end of the last stored value (expanding the stack). These memory accesses have the ESP as the base address, and don not modify the memory regions of the parameters.
- **Local Variable Access.** All memory accesses to the local variable segment in the stack are ignored because there are only two regions where the arguments can be stored:
  1. In the **heap**, if the memory region of the argument is dynamically allocated.
  2. In the **arguments region** (above the EBP), which is not part of the local variables segment.

The mentioned rules are graphically shown in the decision flow presented in Figure 3.5.

Runtime Memory Data Storage (RMDS) . The purpose of this module is to temporarily store the data of the memory accesses filtered by the MAF module, during the execution of the kernel. At the end of the kernel execution, the data is processed by the Permanent Memory Data Storage module and the data in the RMDS module is invalidated. We based this module in the QUAD toolset [28]. The authors propose an efficient Memory Access Tracing (MAT) module for the storage and retrieval of data associated with memory addresses. The MAT module is part of the QUAD toolset.

For each execution of the kernel, two datasets are stored in the RMDS module, the data contained in the memory regions of the parameters before it is modified by the function (*input dataset*), and after probable modification by the function (*output dataset*). As a result, one *trie* data structure is used in the RMDS module, the *Memory Data trie*, which stores the *input dataset* and the *output dataset*. The procedure to store the *input/output datasets* is as follows:

We search in the *memory data trie* for each memory address filtered by the MAF module. If we cannot find the address, it means that it has not been accessed during the current execution of the kernel. Therefore, we store the data present in the memory location in the input and output data blocks.

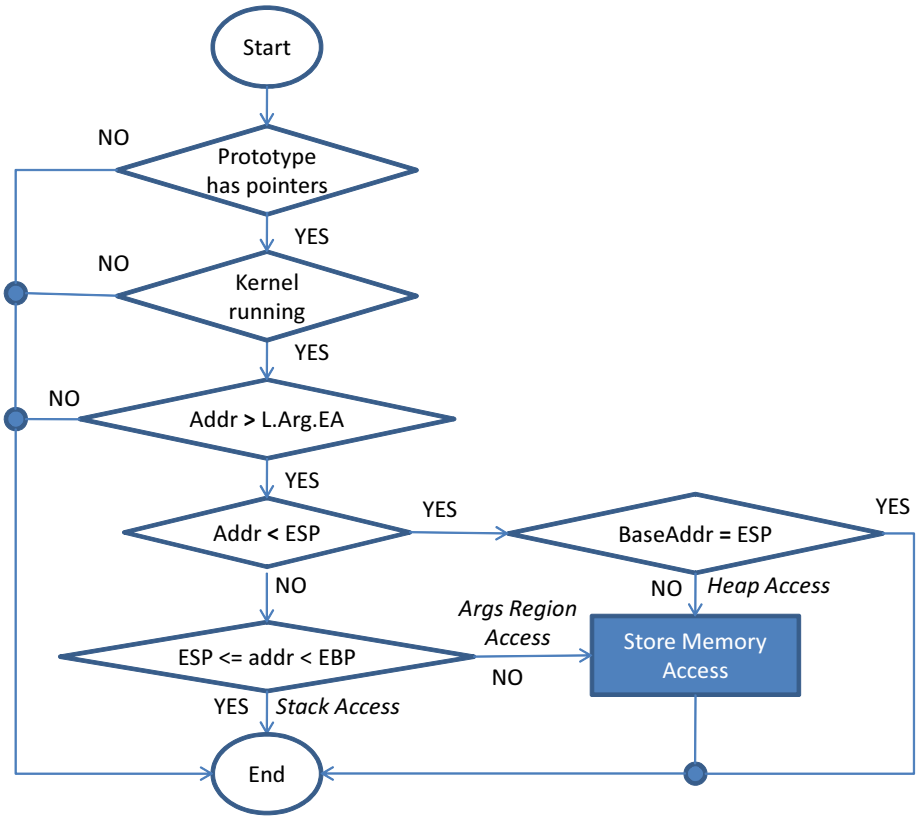


Figure 3.5: Decision Flow of Memory Access Filter module in ARGS tool.

If we can find the address in the *memory data trie*, it means that this address has already been accessed in the current kernel execution. Therefore, we need to determine if a read or write instruction triggered the memory access, because only after a write operation the output data block is updated.

However, we need a pre-processing step in order to determine if the instruction that triggered the memory access would modify the memory address. We can use Pin to determine if the instruction is a read or write. However, Pin does not provide the data to be written in memory until the write instruction is executed. On the other hand, after the execution of an instruction Pin can only provide the address of the instruction executed. Therefore, before executing a write instruction we store in the *Instruction Address trie* the address of the instruction and the EA to be modified. Then, after the execution of any instruction we search in the *Instruction Address trie* the address of the instruction, if the address exists in the trie it means that this instruction is a write instruction, stored previously, and we can get the EA modified. Using now the modified EA and the size provided by Pin we filter the useful memory accesses using the MAF module and finally, the RMDS module stores the data modified in the output data block.

Permanent Memory Data Storage (PMDS) After the kernel is executed, the PMDS module traverses the *memory data trie* contained in the RMDS module to associate the stored data with each of the arguments passed to the function. This module produces two output files one containing the *input dataset* and another one with the *output dataset*. Both files contain continuous blocks of memory associated with the arguments of the function. In order make the memory blocks continuous, we inserted zeroes in the memory regions that were accessed (R/W) during the execution of the kernel.

3.1.2 Evaluation criteria

In order to characterize power consumption of a function using metrics, we need to *find the high level abstraction parameters that can be associated with the power consumed by a function, while reducing prediction time and maintaining the required accuracy.* The required accuracy is defined by the purpose of the power or energy prediction, for instance, in early design decisions (specifically partitioning process) a lot of predictions are required, thus a reduced prediction time is preferred and a prediction that can provide qualitative data for comparison is required. Therefore, the absolute accuracy of the prediction it is not the main concern.

For this thesis, we foresee two types of parameters that can be used for power characterization:

1. Parameters that describe a C-code function statically, meaning that no information regarding the dynamic behavior of the function is described by using these parameters. For instance, the **Software Complexity Metrics** (SCM) that describe some characteristics of a software code, such as, the number of basic blocks, the number of local variables, or the number of loads and stores to memory can be used. The main characteristic of these parameters is that no dynamic profiling is required.

2. Parameters that describe the dynamic behavior of the function. As an example, for the GPP model, the average number of cycles taken by the function call, or the input data passed to a function through the arguments, can be used. These parameters share the characteristic that we need dynamic profiling to obtain them.

In the rest of this thesis, we refer to the parameters selected in this step as metrics. In Section 3.1.2.1 we describe the static metrics that we chose for characterization, and in Section 3.1.2.2 we present the dynamic metrics. It is important to mention that the set of metrics described in the following subsections are not the final metrics used to build the model. Later in Section 4 we analyze the power data obtained during power characterization and define a final set of metrics that have higher correlation with the power consumption.

3.1.2.1 Static Metrics

In [22], the authors select a set of SCMs to build models that predict hardware metrics of FPGAs. We based our work in [22], therefore, we use the same SCMs. The SCMs

and a brief description of each metric, are presented in the appendix 6.1 in the Table 6.1.

3.1.2.2 Dynamic Metrics

Along with the SCMs, we select the amount of data transferred to the function through the arguments on each execution, for characterization. Specifically, this metric refers to the memory regions of the arguments consumed by the function, measured in bytes. Each memory regions is bounded by the address passed through the argument (lower bound) and the last consequent memory address accessed by the function.

Also for each model we can use specific dynamic information of the underlying hardware for characterization. As an example for a GPP, we can use the number of instructions committed, and the number of cycles that the function takes to execute, the number of loads and store. This information should be extracted when the function is executed in the underlying hardware. Specifically for our GPP model, we use an ARM simulator [26] and the simulation results provide the information that is utilized for power characterization.

Table 3.1 also provides a list of dynamic variables selected for power characterization along with a brief description.

Metric	Description	Target Model
data_consumed	Data consumed by a kernel function	GPP/FPGA
sim_cpi	Cycles per instruction	GPP
sim_cycle	Total simulation time in cycles	GPP
sim_ipc	Instructions per cycle	GPP
sim_num_insn	Number of instructions committed	GPP
sim_num_refs	Number of loads and stores	GPP

Table 3.1: Initial set of dynamic metrics

3.2 Experimental Setup

In this section we describe the implementation details of the modelling methodology proposed in this thesis. In general, the experimental setup is composed of a Unit Under Test (UUT) which accepts inputs (parameters) and produce outputs (power data) after an action is performed. The UUT can be any processing unit, such as, a GPP or an FPGA, and it is possible to use a simulation framework or physical measurements on a chip to obtain the power values. During the modelling process, the UUT is considered as a black box in order to make the experimental setup scalable by allowing the exchange of different UUTs.

For this thesis, two different category of experimental setups were designed and implemented. For each category at least one experiment was designed, as explained in Section 3.1. In Table 3.2, we present a comparison of the main characteristics of each experiment. For the experiments presented in Table 3.2, we use the design tools detailed in Table 3.3.

The experimental setup that uses physical measurements on a chip can provide more accurate values of power consumption and, as such, the quality of the prediction models.

However, this approach proved to be time consuming and a non-trivial task, because it requires high-end measurements tools.

Alternatively, the experimental setup with simulation frameworks is useful to prove the modelling methodology proposed in this work. It allows a wider exploration of hardware parameters that affect the power and energy consumption in a UUT. As an example, the Sim-Panalyzer simulator [26] used in our experiments allows changing the clock frequency, the number of cache levels, the cache memory size, and other architecturals parameters of a GPP, which can be included in the modelling process to investigate its corelation with power or energy consumption.

In Section 3.2.1, we present the implementation details of the experimental setup using physical measurements. It is important to clarify that the results using physical measurements were not satisfactory, because the used measurement tool does not have the required technichals characteristics to accurately measure the experiments designed. However, we present the details of the measurements and provide relevant discussions of the results.

In Section 3.2.2 we present the implementation details of the experimental setup using the simulation frameworks presented in Table 3.2.

3.2.1 Physical Measurements

The experimental setup category that uses physical measurements aims to detect slight differences in current and voltage, caused by the execution of specific actions in a UUT. Throughout this thesis we call these actions as, Action Under Test (AUT). The start and end of the AUT is controlled and monitored in order to measure only the AUT. The AUT is a designed experiment which accepts modification of its dynamic characteristics using the parameters selected for modelling.

As an example, with a GPP as the UUT we can use the ISA as the abstraction level, and the bit-switching activity of the operators as a parameter that characterize the power consumed by an instruction. Then, an AUT would exercise different values

Experiment	PowerPC.Virtex5.GPP.A.1	StrongARM.SimPanalyzer.GPP.B.1	VIRTEX5.Xpwr.FPGA.B.2
UUT	GPP PowerPC 440	GPP StrongARM	FPGA Virtex 5 XCVFX130TFF1738
Input of the model	ISA	Functions in C code	Functions in C code
Measurement method	Physical measurements: using SysMon on-chip module	Simulation framework: Sim-Panalyzer 2.0.3	Simulation framework: Xilinx *xpwr* tool
Clock frequency	475 MHz [41]	200 MHz	100 MHz
Power supply voltage	1 v	1.8 v	1 v

Table 3.2: Comparison of two experimental setups implemented in this work.

Design tool	Version
Sim-Panalyzer	2.0.3
Xilinx ISE suite	12.2
Xilinx Chipscope	11.2
Command line tools of Xilinx	12.2
ModelSim	6.5
DWARV C-to-VHDL compiler	r945M

Table 3.3: Design tools used in the experiments of this thesis

of bit switching activity of the operators, to allow the power characterization of the switching activity. As a result, it is necessary that the AUT allows the change of the switching activity of the operatiors in an instruction.

As a result, the AUT should admit the change of bit switching activity of the operators.

In this experimental setup the principal concern is the accurate measurement of power consumption. This problem can be addressed in many different ways, and the choice depends on the available resources and the UUT to be measured. This research is conducted in the context of DWB, and uses the MOLEN architecture [39] as the chosen heterogenous architecture. MOLEN is implemented in a Virtex 5 Chip (XC5VFX130TFF1738-2) that contains reconfigurable fabric and two PowerPC 440 cores embedded in the same chip.

We present a detailed description of the available resources for this experimental setup as follows.

- The PowerPC 440 cores and the FPGA fabrics are enclosed in a single chip, as a result, we cannot measure independently the power consumed by each UUT (the GPP and the FPGA). Therefore, we measure the power consumption of the entire chip and carefully design experiments that isolate the power of each UUT.

- The Virtex 5 device has an on-chip hardware module composed of Analog to Digital Converters (ADC) called System Monitor (SysMon). SysMon is able to measure the voltage, and the current of the power source of the chip. The current is measured indirectly by measuring the voltage drop in a Kelvin resistor that is in series with the power source.

- The Virtex 5 chip is placed in a ML510 development board which already integrates the Kelvin resistor with a value of 2.2 mΩ. The ML510 board provides also two test points that allow access to the terminals of the Kelvin resistor for external measurement.

- Xilinx provides the tools (ChipScope) to control SysMon and get the power measured, using a JTAG I/F. Moreover, Xilinx provides a Tcl interface which can be used for an automated solution.

Apart from the available resources summarized above, there are some requirements which were considered for this experimental setup:

- The number of tests to be made is high, we want to measure the different permutations between the 71 instructions of the PowerPC ISA. Consequently, the experimental setup should be automated and allow unattended execution of the tests.

- We want to measure power consumption only during during the execution of the AUT. Hence, the experimental setup should be able to precisely start and stop a measurement.

Since the main element of this experimental setup is the instrument used to measure the power consumption, we provide a detailed description of the technical characteristics of SysMon. This description is provided in the following section.

3.2.1.1 System Monitor (SysMon) technichal characteristics

The SysMon is a built-in hardware module in the Xilinx Virtex 5 chip which can measure the following characteristics:

- The voltage in the power sources of the chip (Vcc and Vccaux).
- The chip temperature, provided in C.
- Voltage off-chip with 16 ADCs.
- Differential off-chip voltage using the dedicated VpVn channel.

All the ADCs integrated in SysMon have a sampling frequency of 200 Ksamples per second. When the SysMon is enabled, it provides the measured values in a set of registers available through the JTAG interface.

The SysMon does not integrate a way to measure current of the Vcc power source directly. However, with the ML510 board, the setup to measure current of Vcc consists of VpVn inputs connected to a 2.2 mΩ Kelvin resistor that is connected in series with Vcc. The VpVn channel can measure differential voltage between Vp and Vn inputs. As a result, we can indirectly measure the current of the power supply (Vcc) using Ohms law.

Further details of the resolution of the sensors of SysMon are as follows.

Resolution of the power supply voltage sensor According to the Xilinx documentation, the Virtex 5 integrates an on-chip power supply sensor that allows measurement of the voltage in the power supply. The sensor samples and attenuates the voltage of the power supplies ,Vccint and Vccaux, by a factor of three. The range of measurements are in the range of 0 volts to 3 volts with a resolution of one least significant bit (LSB) = 2.93 mV. Equation 3.1 describes the transference function of the power supply sensor:

$$SupplyVoltage[volts] = \frac{ADCcode \times 3}{1024} \tag{3.1}$$

The output code of the sensor is 10-bit long, therefore, the digitized data of the sensor can present up to 1024 different values, with a range from 0 up to 0x3FFh.

Resolution of the power supply current sensor A 2.2 mΩ resistor is connected in series with the Vcc power supply, and VpVn inputs are used to measure the voltage drop in the resistor, which indirectly give us the current.

The VpVn inputs are connected to an ADC with 10-bit output code, configured in differential mode. In this mode the VpVn ADC provides values from -500(0x200h) to 499(0x1FFh), with one LSB = 0.977 mV.

Now that we know the resolution of the VpVn ADC, we can obtain the resolution of the current measurements using Ohm's law. The resolution of the current measurements is shown in Formula 3.2.

$$I_{LSB} = \frac{VpVn_{LSB}}{R} = \frac{0.977mV}{2.2m\Omega} = 443.892mA \tag{3.2}$$

3.2.1.2 Experiment PowerPC.Virtex5.GPP.A.1 description

Based on the presented conditions and requirements, we have designe the Experimental Framework shown in Figure 3.6. The experimental setup consist of two parts, the Power Measurement Tool (PMT) and an accompanying set of bash scripts. These scripts drive the Xilinx tools, such as Chipscope and Impact, to get data from SysMon and program the board, respectively. Furthermore, the interface with the PMT. This tool, which is written in Java, orchestrate the execution of the experiment and gathers the data. The main features of PMT are:

- It provides remote access to the PC connected to the board, using an open source SSH library (Remote Connector module).
- The AUT can be scheduled using First In First Out(FIFO) scheduling policy (Test Scheduler module).
- It has a Graphical User Interface (GUI) to upload the AUT to the scheduler. For each AUT, the user has to provide a description, a name, and a bit file required to program the AUT in the board.

The Figure 3.6 shows the software architecture of the PMT within the experimental framework.

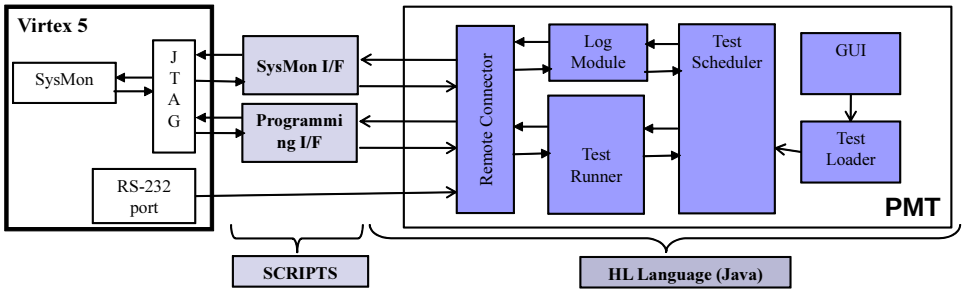


Figure 3.6: Experimental Framework for experiment PowerPC.Virtex5.GPP.A.1

3.2.1.3 AUT description

. In the Virtex 5 FPGA we require the following two files to program the board:

1. A bitstream that contains the information to configure the reconfigurable fabrics and HW modules available in the FPGA. This file is generated with the Xilinx design tools.

2. An object file in the Executable and Linkable Format (ELF) that contains an executable image which is executed in the PowerPC processor.

Both files are wrapped in a single file (*download.bit*) that can be loaded to the Virtex 5 to program the board. The startup process, after loading the *download.bit* file, starts with one PowerPC Core booting, after the booting procedure finishes, the *main* function is executed. When the main functions starts execution, the control flow is given to the program stored in the ELF file. For our experimental setup, after the control flow is given to the main function we disable the cache memory and enable the UART of the PowerPC, and then the AUT is started.

For the FPGA, the reconfigurable fabrics are configured using the bitstream. The functionality and the control flow depend entirely on the design implemented. For the MOLEN architecture, the PowerPC acts as the master component in the Processor Local Bus (PLB). The Custom Computing Unit (CCU) is an slave in the PLB, therefore, for the FPGA and the GPP the PowerPC will start the AUT.

We present the implementation details of the AUT for the GPP as follows.

Description of the AUT in the GPP. For the GPP we select the ISA of the PowerPC as the abstraction level, similar to the work presented in [38]. Based on the abstraction level chosen we defined a template for AUT which is composed of the following elements:

- An ELF file which contains the main function with a loop that executes a specific number of times a single assembly instruction of the ISA.
- A BIT file that contains the configuration information to implement MOLEN architecture in the Virtex 5 FPGA.

The pseudo-code presented in Listing 3.1, sketches the template in C code that was used to create the ELF files for power characterization:

Listing 3.1: Template to create ELF file for PowerPC

```
1  int main() {
2      init_platform(); // Disable caches and enable UART port.
3      signal_start(); // Send character through serial port
4                      //to signal start of AUT.
5      while ((--iterations)>0) {
6                      loop();          // Loop of one ISA instruction
7                      //repeated multiple times.
8      }
9      signal_end();    // Send a character through serial port
10                                      //to signal end of AUT.
11 }
```

The *loop( )* function inlines the assembly instruction to be repeated multiple times. The average number of instructions that are inlined in a *loop* is 500. And the number

of repetitions in the outer (*while*) loop is 40 million. Therefore, one single instruction is executed on average 20×10^9 times per AUT.

The code in Listing 3.2, shows an example of a *loop* function for an *add* instruction. As can be seen in the code, the *add* instruction uses registers to transfer data, as many other instructions. Therefore, we tried to randomize the data passed to the instructions by changing the order of the registers used to transfer data, with the purpose to account for the effects of the registers, in the power consumption. In the Listing 3.2 we can observe the randomization method we are referring to.

Listing 3.2: Example of an AUT in the PowerPC

```
1  #define loop() __asm__ __volatile__(\
2          "add %%r4, %%r3, %%r2\n"\
3          "add %%r3, %%r4, %%r2\n"\
4          ....
5          "add %%r2, %%r3, %%r4\n"\
6          "add %%r2, %%r2, %%r4\n"\
7          : / no outputs /\
8          : / no inputs /\
9          : "%r3", "%r2","%r4" \
10         );
11 #endif
```

Following a similar structure as the one shown in Listing 3.2, a set of loops were created to measure the power of all the instructions in the ISA of the PowerPC.

Finally, in order to measure the power consumption of each instruction, we send a character through the serial port to signal the start of the AUT. Since the serial port is monitored by the PMT tool, when it receives the *start* character, it begins the measurements, using SysMon. When the AUT ends, it sends a character to the PMT tool to stop the SysMon measurements. All the measurements made by SysMon are obtained through the JTAG interface and does not affect the operation of the GPP.

Notice that the results and analysis presented in Section 4.1.1, were not satisfactory. We will present that the results obtained with SysMon present a non-repeatable behavior. In order to root the problem of this behavior, we perform a validation experiment that uses an oscilloscope to measure the voltage dropped in the Kelvin resistor. This experiment is referred as PowerPC.Virtex5.GPP.A.1.4 during this thesis, and the characteristics of the experiment are presented as follows:

1. The same bit files used during the experiments with SysMon were used in these experiments.

2. Since the measurements made with the oscilloscope were completely manual, we did not test all the instructions measured with SysMon.

3. We add two instructions in this experiment, the *lbz* and *machhw* instructions, in order to include more types of instructions.

4. We add a new experiment that executes an entire application, the CCU-addition. In this application the PowerPC executes a loop of additions using an adder implemented as a CCU of the MOLEN architecture. This experiment is a modified version of a validation test provided in the implementation of MOLEN in the Virtex 5. The only change we add was disabling the output of the results through serial port.

5. The test CCU-add-serial was included, this test is similar to the test CCU-addition but the results were transferred using the serial port. This experiment is a validation test provided in the implementation of MOLEN in the Virtex 5.

This behavior was rooted to, the inadequacy of the sampling rate of SysMon required by the signal to be measured.

Conclusions . Based on the results and analysis presented in Section 4.1.1, we observed a non-repeatability of the results obtained with SysMon. This behavior was rooted to, the inadequacy of the sampling rate of SysMon required by the signal to be measured. As a result, we did not use the results of this experiment to create a model, however, as we mention at the beginning of this section, the success of making a model, based on physical measurements, depends on the available resources. Therefore, based on the experience obtained in this experiment we provide the following general conclusions:

1. It is important to know the characteristics of the signal to be measured, in order to decide whether the available instrumentation is able to measure the signal with the required accuracy.

2. The current complexity of MPSoC, such as the Virtex 5, provides a new challenge for modelling power with data obtained from physical measurements. The main problem is measuring the (sligh) power consumed in the chip, caused by an AUT, because the percentage of logic used for the AUT is becoming smaller in comparison with all the logic inside the chip.

3. The lower the abstraction level of the AUT, the more difficult its to measure its power consumption. Two approaches can be used to solve this problem, either select high-end instruments to perform the measurements, or increase the abstraction level of the AUT to be measured.

4. If a high-end instrument it is chosen as the solution, the sampling rate of the instrument is an important characteristic used to select the right measurement tool to model power consumption. The reason is that current MPSoC can work at high clock frequencies, therefore an instrument with a high sampling frequency would help to improve the repeatability of the power measurements, by getting more samples of the AUT.

5. In this *emperimental framework* we did not include temperature measurements but in the deep submicron (DSM) technology, the temperature is becoming an important factor that affects power consumption, therefore, it is important to

include it in the modelling methodology in order to reduce the non-repeatability of the power measurements between experiments.

6. The experimental setup should provide a high controllability to start and stop the measurements in the shortest possible amount of time in order to reduce the disturbances caused by the measurements of activities different than the AUT.

3.2.2 Simulation framework

In this section we present the *experimental framework* that uses a simulation framework as UUT. Figure 3.7, presents the modelling methodology when a simulation framework is used as UUT. The *experimental framework* proposed in this section has the following advantages:

- A simulation framework provides the opportunity to fully automate the experimental setup. As a result, we can explore different architectural parameters that affect power consumption by configuring the simulation framework, or even changing the UUT.
- A simulation framework provides the opportunity to validate the modelling methodology proposed, and to save time in the implementation because we do not need complex physical measurents.

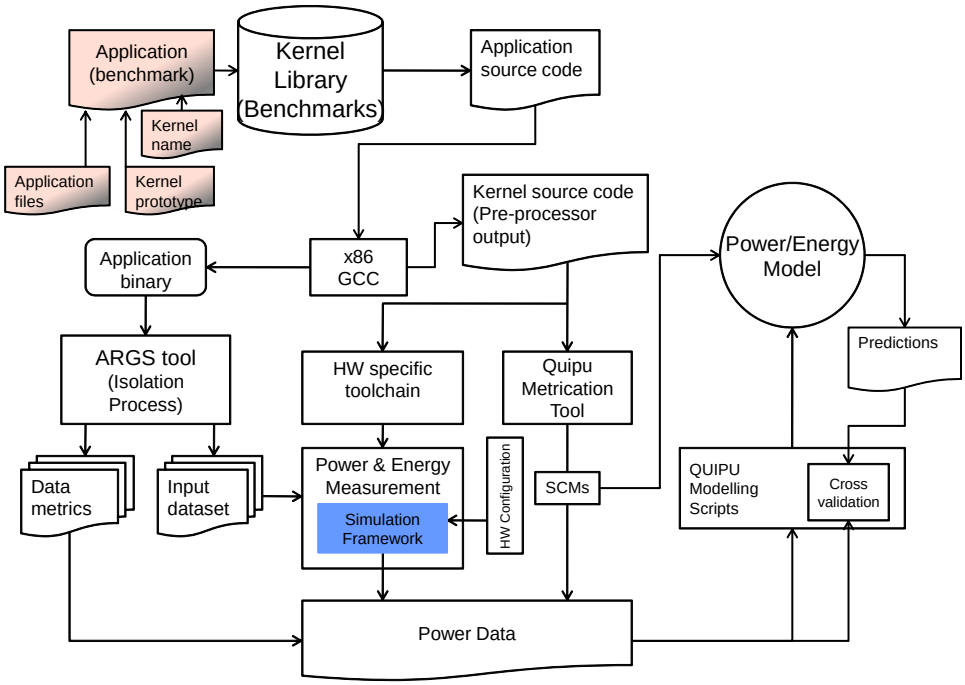


Figure 3.7: Experimental setup using simulation frameworks

However, one of the main disadvantages of this experimental framework *is that the accuracy of the high level models is bounded by the accuracy of the simulation framework.*

In previous sections we have discussed the implementation details of some modules present in the modelling methodology. Therefore, in Section 3.2.2.1 we focus on the general implementation of the automated experimental setup. The implemented *experimental framework* has the flexibility of changing the UUT. However, it requires one-time modifications to build a new model for a different UUT. Therefore, in Section 3.2.2.3, we present the changes required to use the Xpower tool of Xilinx, to create a model for a Virtex 5 FPGA. In Section 3.2.2.2, we present the changes required to use an ARM simulator as UUT for the GPP model.

3.2.2.1 Description of the automated experimental

The experimental setup is implemented in a Linux machine with Suse OS and x86 processor. Therefore, we choose the *bash* scripting language and *make* utility to automate the experimental setup. The design goals of this automated experimental setup are:

- **Scalability**. We consider important to provide an easy way to include more applications (with kernels) in the kernel library.
- **Flexibility**. This experimental setup should allow an easy one-time modification to include a different UUT.

In order to achieve the goal of **scalability**, we consider each application as a plug-able module, which is implemented as a folder that contains the source code of the application, and a folder called PowerModel. The PowerModel folder contains the following information required by the experimental setup:

- **Kernel file**. This is a plain text file that contains, on one line, the name of the function to be measured and its prototype. As a suggestion this file should have the extension *.kernel to allow an easy identification in the PowerModel folder. The following format must be used to define the function name and prototype: *function_name:arg1_type:arg2_type:arg3_type....* To define an argument passed by reference the user has to include the asterisk symbol (*) after the type. As an example, to define an integer passed as reference the user would use: *kernel_name:int*....*
- **Application file**. This is a plain text file that contains on one line the C-files that compose the application, each file has to be separated by a space. In the current implementation, the name of the application file should match the name of the application, because this information is used by the scripts that obtain the power and provide results.
- **Metrics file**. This is a plain text file that contains on one line the name of the file where the kernel is defined. The source code file defined in the metrics file might require pre-processing by the gcc and maybe some manual modifications to allow the *qpm-metricator* tool, presented in [22], to extract the SCMs.
- **Cflags file**. For a UUT that requires cross-compilation of the source code, defined in the *application* file, we define in this plain text file the compilation flags required to compile the application.

The **flexibility** in the experimental setup is achieved by programming in *bash* scripts the one-time modifications required for a new UUT. The actions that these scripts perform depend on the UUT, but in general can be described as follows:

- **Pre-process the kernel source code**. The pre-processing includes actions such as, creating a header file for each *input dataset* provided by the ARGS tool (for a GPP model), or translating the kernel C source code to VHDL, using a C-to-VHDL translator (for an FPGA model).
- **Execution of the UUT toolchain**. The required actions depend entirely on the toolchain provided by the UUT vendor, for instance, for a GPP model an action is the cross-compilation of the source code, but for an FPGA model, the actions include synthesis, mapping, and place & route.
- **Power Measurement**. This action depends on the simulation framework being used as UUT.
- **Power data consolidation**. In this step, the power data obtained for an specific UUT is consolidated in a single file, along with the SCMs of the kernel and the *data metrics* of the *input datasets* provided by the ARGS tool.

These scripts use the information stored in the PowerModel folder, such as the *kernel* file, and the *application* file. The scripts are stored in a folder called PowerModelBin that is exported in the Linux *path* to be used by any *application module*. As a result, any modification in the scripts will affect all the applications in the *kernel library*.

Finally, in order to automate the measurement of power with the experimental setup, we use the *make* command of Linux. For each application folder, we include a *Makefile* which contains a *'recipe'* with the actions outlined above. In order to provide an easy way to include more kernels to an application, and more applications to the *kernel library*, the makefile has to include a set of general rules defined in another makefile, that is stored in the PowerModelBin folder.

In Listing 3.3 we include some examples of general rules that apply to all the applications. Notice that these general rules use *make* variables which are defined by each application.

Listing 3.3: Examples of general rules defined for all applications in the kernel library

```
1  get_gpp_power:
2      process_folders.sh  (KERNEL_FILE)  (APP_FILE)  (CFLAGS_FILE)
3  process_gpp_statistics:
4      process_statistics.sh  (KERNEL_FILE)  (APP_FILE)
5  ....
6
7  app2vhd:
8      app2vhd.sh  (APP_FILE)
9  kernel2ngc:
10     kernel2ngc.sh  (KERNEL_FILE)
11 create_kernel_tb:
12     create_kernel_tb_ngc.sh  (KERNEL_FILE)
13 get_fpga_power:
14     get_power_kernel_fpga.sh  (KERNEL_FILE)
```

```
process_xpwr_logs:
    process_kernel_xpwr_out.sh  (KERNEL_FILE)
....
```

In Listing 3.4 we present the template of a Makefile which has to be included in the PowerModel folder of each application, we use as example a kernel called *bytesum*.

Listing 3.4: Template of a Makefile for an application in the kernel library

```
include  (PM_BIN)/power_model.make

BYTESUM_FILE=bytesum.kernel
POLYNOMIALBIN_FILE=polynomialbin.kernel
FUNCTIONS_APP_FILE=functions.data
FUNCTIONS_CFLAGS_FILE=cflags.data
FUNCTIONS_METRICS_FILE=functions_metrics.data

bytesum_get_gpp_power: eval_bytesum get_gpp_power
bytesum_process_gpp_statistics: eval_bytesum process_gpp_statistics
bytesum_kernel2ngc: eval_bytesum kernel2ngc
bytesum_create_kernel_tb: eval_bytesum create_kernel_tb
bytesum_get_fpga_power: eval_bytesum get_fpga_power
bytesum_process_xpwr_logs: eval_bytesum process_xpwr_logs

eval_bytesum:
         (eval KERNEL_FILE= (BYTESUM_FILE))
         (eval APP_FILE= (FUNCTIONS_APP_FILE))
         (eval CFLAGS_FILE= (FUNCTIONS_CFLAGS_FILE))
         (eval METRICS_FILE= (FUNCTIONS_METRICS_FILE))
```

3.2.2.2 Experiment the VIRTEX5.Xpwr.FPGA.B.2

The required one-time modifications to create this experiment are presented in this section. This experiment was designed in the context of the DWB, and as such we use the MOLEN architecture and implement each kernel as a CCU. We also use the BlueBee toolchain [6] which simplifies the development in heterogeneous architectures using the Molen machine organization. For a Virtex 5 FPGA the BlueBee toolchain uses the Xilinx tools during the design flow.

The actions performed in this experiment are listed below:

1. **C-to-VHDL tranlation**. For each kernel in an application we tranlate the C function to VHDL. For this purpose we use the DWARV tool [42]. The output of this stage is a VHDL translation of the kernel.

2. **Design a synthesis testbench**. Since we only want to measure the power consumption of the kernel function, we use the CCU wrapper entity defined in the implementation of Molen for Virtex 5. The CCU wrapper is a generic entity that only contains the input and output ports of a CCU, and instantiates a CCU as a black-box. The CCU wrapper can be instantiated and synthesized without defining the CCU black-box, and later in the power measurement stage we can substitute the black box for the desired CCU and perform the measurements. The output of this stage is a netlist (ngc file) with an instance of the CCU wrapper.

3. **Design a simulation testbench**. We design a testbench that provides all the required signals to the CCU wrapper, including the clock signal, and the *input dataset* of each kernel.

4. **Synthesis & Translation**. For each kernel in the *kernel library*, we use the BlueBee toochain to synthesize the CCU and obtain a netlist. Then, we merge the obtained CCU netlist with the synthesis testbench using the ngcbuild tool of Xilinx. The output of this step is a netlist that contains a CCU wrapper as the top module, and the required CCU.

5. **Mapping**. The netlist obtained from step 4 is mapped in a Virtex 5 FPGA, this step using the *map* tool of Xilinx.

6. **Place & Route**. The design mapped in step 5 is placed and routed in this step using the *par* tool of Xilinx.

7. **Power measurement**. Using the simulation testbench and a post place & route simulation model of the CCU wrapper, we simulate the design with Modelsim to obtain a Value Change Dump (VCD) that contains information of the switching activity of the signals in the design. The VCD file is used along with the placed & routed CCU wrapper to obtain the power consumed, using the xpwr tool of Xilinx. For each *input dataset* provided by ARGS tool we get a power sample which is stored in a file.

8. **Power data consolidation**. We consolidate in a single file the power samples obtained in the previous step, the metrics of the kernel (obtained with the qpm-metricator tool), and the data metrics of each *input dataset* used during the measurements.

3.2.2.3 Experiment StrongARM.SimPanalyzer.GPP.B.1

In this experiment we use the sim-panalyzer [26], a SimpleScalar ARM power simulator. This simulator allows the configuration of a set of architectural parameters, such as, levels of cache, size of cache, and clock frequency. This configuration is given to the simulator in a configuration file, which also includes the effective capacitance of some functional units defined in the processor. In this experiment we use a configuration file provided by the sim-panalyzer project that describe the characteristics of a StrongARM processor. All the parameters of the configuration file are presented in Section 6.2, however, the most relevants ones are presented in the Table 3.4.

Parameter	Value
Clock frequency	200 Mhz
Source voltage	1.8 v
Instruction execution	In-order
Cache levels (for instrucions and data)	2

Table 3.4: Architectural parameters used in sim-panalyzer

In order to execute an application in the sim-panalyzer we compiled a cross-compiler for the ARM processor.

Finally, the actions programmed in the scripts for the ARM processor are listed below:

1. **Create header file**. In order to pass to the kernel the *input datasets* generated with ARGS tool, we use a script that generates a header file. This header file contains an *input dataset* and is included in the source code of the application to be executed in the sim-panalyzer. This script can handle functions with different size and type of parameters using the prototype defined in the *kernel file*.

2. **Source code modification**. The application to be executed in the sim-panalyzer contains only a main function that includes the header file of step 1, and the kernel under test. For this purpose, the script searches for the *main* function in the application source code to disable it. It then creates new version that contains only the kernel under test.

3. **Cross-compilation**. Using the ARM linux cross-compiler, we compile the modified application that only executes the kernel under test.

4. **Power measurement**. In this step, the sim-panalyzer is executed using the cross-compiled application and the configuration file defined for the StrongARM. For each *input dataset* obtained with the ARGS tool we create a header file, modify the source code, cross-compile the application, and then execute it in the simulator. This provides a power sample per each *input dataset*, all the power samples are stored in a file.

5. **Power data consolidation**. We consolidate in a single file the power samples obtained in the previous step, the metrics of the kernel (obtained with the qpmmetricator tool), and the data metrics of each *input dataset* used during the measurements.

4 Results and Analysis.

In this chapter we present the results of the experiments performed in the context of this thesis. For each experiment we use a set of kernels contained in the *kernel library*, the description and details of the *kernel library* are presented in the Table 6.2 in the Appendix 6.4.

The experiments performed in this thesis can be categorized as follows:

A. **Physical Measurements**. In this category, we measure the power consumed by a kernel using physical measurements on the UUT. One experiment was designed in this category, and is described as follows:

   1. **Experiment PowerPC.Virtex5.GPP.A.1**. In this experiment we physically measure the power consumed in a PowerPC processor, that is embedded in a Virtex 5 chip. The results of this experiment are presented in Section 4.1.1

B. **Simulation framework**. In this category, we use a simulation framework to measure the power consumed by a kernel during its execution in the UUT. In this category we designed two experiments that are summarized as follows:

   1 **Experiment StrongARM.SimPanalyzer.GPP.B.1**. This experiment uses the Sim-Panalyzer simulator with a StrongARM configuration as UUT. The Sim-Panalyzer estimates the power consumption of kernels running in the StrongARM processor. The StrongARM is configured with a clock of 200MHz and a voltage source of 1.8v. The complete configuration file is presented in Section 6.1. The results and an analysis of this experiment are presented in Section 4.1.2.

   2 **Experiment VIRTEX5.Xpwr.FPGA.B.2**. This experiment uses the *xpwr* tool of Xilinx to measure, the power consumption of a kernel running in a Virtex 5 XC5VFX130TFF1738-2 (with the MOLEN machine organization). The results and analysis of this experiment are presented in Section 4.1.3.

In order to assist in the identification of the experiments and to provide a proper structure in this report, we define the following naming convention for the experiments: $Modelling_{HW}.Measurement_{HW}.UUTtype.Category.Experiment\#$. Where, the $Modelling_{HW}$ is the hardware which is being modelled, the $Measurement_{HW}$ is the chip being measured or the simulation framework being used, the $UUTtype$ defines in general for which UUT this experiment is made, the $Category$ is the category of the test, and the $Experiment\#$ is the number of experiment in that specific category.

The results presented in Section 4.1.2 and Section 4.1.3 were used to create a set of models. Therefore, in the Sections 4.1.2.1 and 4.1.3.1, we present the models derived from the modelling step in each experiment.

4.1 Experimental results

In this section, we present a summary of the results obtained in the following experiments:

- **Experiment PowerPC.Virtex5.GPP.A.1**.
- **Experiment StrongARM.SimPanalyzer.GPP.B.1**.
- **Experiment VIRTEX5.Xpwr.FPGA.B.2**.

These experiments were designed to provide an answer to the research questions presented in Chapter 2. A summary of the questions is presented as follows, for reference:

- *Would an automatic high level modelling methodology help to investigate the different parameters that affect power consumption?*
- *Are the low level details of a HW processing unit necessary during the modelling process? Does considering the HW processing unit as a black-box provide more flexibility and scalability to the methodology?*
- *Does using a HLL (C code) as input for a prediction model reduce the prediction time of the model? Do the predictions obtained with this model provide qualitative and quantitative data useful for the partitioning process in the DWB design flow?*

4.1.1 Experiment PowerPC.Virtex5.GPP.A.1

The main characteristics of this experiment are summarized in the Table 4.1.

Parameter	Value
UUT	PowerPC440 GPP (embedded in a Virtex 5 chip)
Input of the model	Instructions of the ISA
Measurement method	Physical measurements: SysMon on-chip module
Clock Frequency	475 MHz [41]
Power supply voltage	1v

Table 4.1: Main characteristics of experiment PowerPC.Virtex5.GPP.A.1

In this experiment, one experiment is performed three times and a different identifier is given to each execution of the experiment, and an extra experiment is done for validation. The identifiers of the experiments are described below:

1. **Experiment PowerPC.Virtex5.GPP.A.1.1**. Experiment performed with SysMon, an on-chip module in the Virtex5 chip. Execution 1.
2. **Experiment PowerPC.Virtex5.GPP.A.1.2**. Experiment performed with SysMon, an on-chip module in the Virtex5 chip. Execution 2.
3. **Experiment PowerPC.Virtex5.GPP.A.1.3**. Experiment performed with SysMon, an on-chip module in the Virtex5 chip. Execution 3.

4. **Experiment PowerPC.Virtex5.GPP.A.1.4**. Experiment performed with an oscilloscope to validate the results of experiments: PowerPC.Virtex5.GPP.A.(1)(2)(3).

In Figure 4.1 we show the results of the first three sub-experiments: PowerPC.Virtex5.GPP.A.1.1, PowerPC.Virtex5.GPP.A.1.2, and PowerPC.Virtex5.GPP.A.1.3. For each sub-experiment, we measure the power consumption of different instructions of the ISA, such as, *'nop'*, *'add'*, *'addc'*, *'addme'*, and *'andc'*. Each instruction was executed on average 20×10^9 times in each experiment, and one ELF file was used in the three experiments for each instruction.

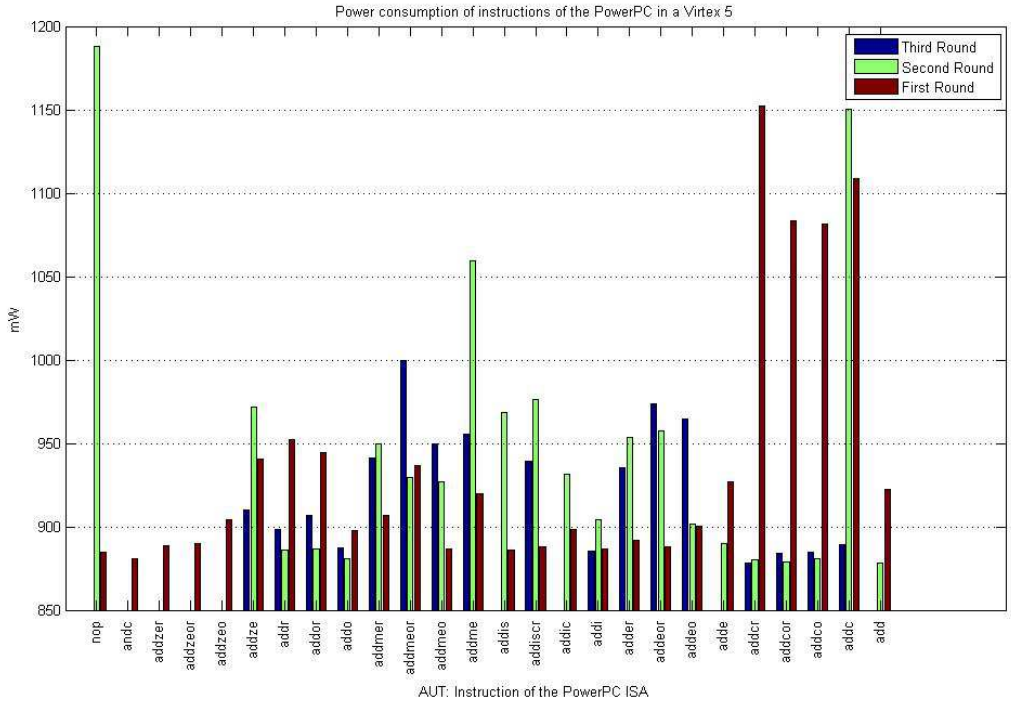


Figure 4.1: Power measurement results of the PowerPC ISA in a Virtex 5

After the execution of different instructions of the ISA, we obtained a set of preliminary results, and the main problem we observed was the non-repeatability between experiments of the power results. As an example of the non-repeatability of the power measurements, the instructions *'nop'*, *'addcr'*, and *'addcor'* show a considerable difference between the sub-experiments results in Figure 4.1.

In order to explain the non-repeatability of the results observed in Figure 4.1, we perform a data analysis of the power samples obtained from experiments PowerPC.Virtex5.GPP.A.1.1 and PowerPC.Virtex5.GPP.A.1.2.

Power distribution. During the execution of each AUT in an experiment, we collected a set of power measurements values from the JTAG. The distribution of the power data of two experiments, PowerPC.Virtex5.GPP.A.1.1 and PowerPC.Virtex5.GPP.A.1.2,

is shown in Figure 4.2. We present the results of these two experiments for the *nop*, *addcr*, *addo*, and *addi* instructions. These instructions were selected because the *nop* and *addcr* instructions have a significant difference between experiments one and two. On the other hand, the instructions *addo* and *addi* have a small difference.

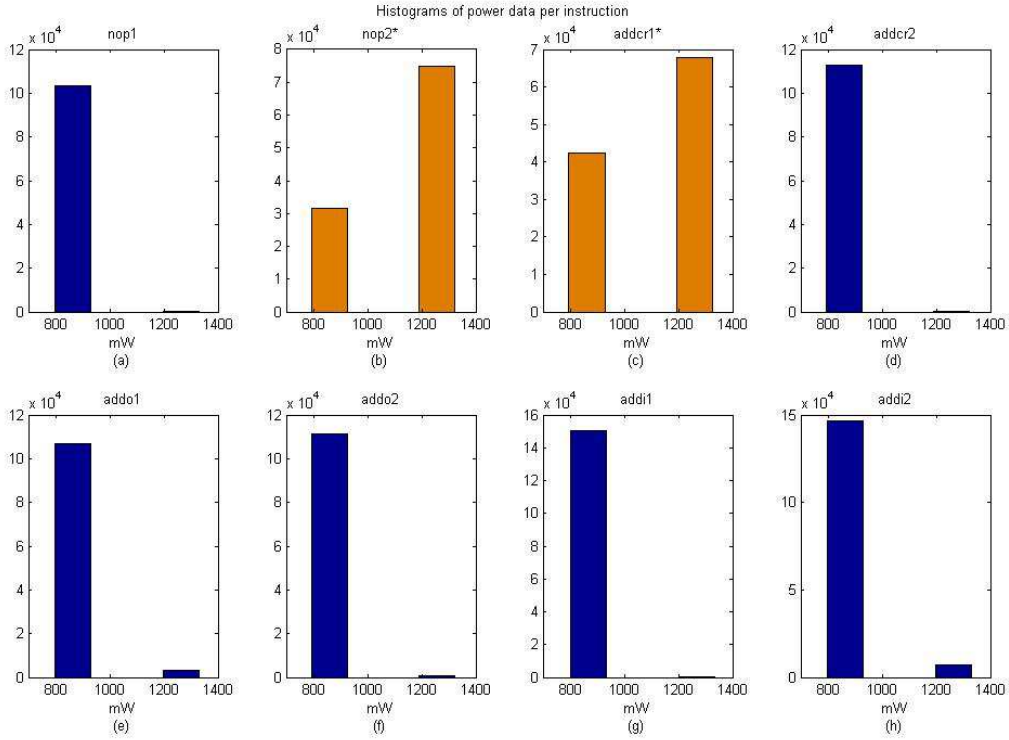


Figure 4.2: Histogram of power data per instruction

Figures 4.2 (a) and (b) show the power distribution for the *nop* instruction, in the experiments PowerPC.Virtex5.GPP.A.1.1 and PowerPC.Virtex5.GPP.A.1.2. Figures 4.2(c) and (d) show the power distribution of measuring the *addcr* instruction, in the experiments PowerPC.Virtex5.GPP.A.1.1 and PowerPC.Virtex5.GPP.A.1.2. Notice that in both intructions one experiment (PowerPC.Virtex5.GPP.A.1.1 for the *nop* instruction) has more samples of power that fall into the group of ∼1300 mW. As a result, its mean value is bigger compared with the other experiment (PowerPC.Virtex5.GPP.A.1.2 for the *nop* instruction). The rest of the samples measured fall into a group of ∼800 mW. Figures 4.2 (e) and (f), show the distribution of results for the instruction *addo* collected in experiment PowerPC.Virtex5.GPP.A.1.1 and PowerPC.Virtex5.GPP.A.12. Figure 4.2 (g) and (h), shows the same for instruction *addi*. In these graphs observe that both experiments have similar distribution of power samples. However, all the values also fall into two groups, one of ∼800 mW and the other of ∼1300 mW.

The distribution of these power samples show a clustering behavior, that can be observed in Figure 4.2. The power samples are clustered in two groups, one ∼1300 mW and another ∼800 mW. This clustering behavior can be explained as follows:

By definition, the power is the measure of how much work can be done in certain ammount of time. Using this definition for electrical power, we can compare the the voltage as the work per unit charge, and the current as the rate at which current is moved through a conductor in one second. As a result, we can use Equation 4.1 to define the power consumption in terms of voltage and current.

$$P = V \times I \tag{4.1}$$

Then, in our chip we know that the voltage does not change during an AUT, because we do not have enabled any voltage scaling feature. As a result, we can observe that the current has a linear relationship with the power consumption and the voltage does not influence the power consumption because its value is constant. This can be confirmed with the histograms shown in Figure 4.3, that show the distribution of voltage, current, and power obtained for the *addcr* instruction in two different experiments, PowerPC.Virtex5.GPP.A.1.1 and PowerPC.Virtex5.GPP.A.1.2.

Voltage and current distribution. In Figure 4.3, we present the power, voltage and current distribution for the *addcr* instruction. These results were obtained from the experiments PowerPC.Virtex5.GPP.A.1.1 and PowerPC.Virtex5.GPP.A.1.2. In Figure 4.3 (a) and (d), we can observe that the voltage is almost constant at $\sim$ 995 mV, with slight variations caused by the power source noise and the instrumentation. The current is presented in Figure 4.3 (b) and (e). Observe that the current values are clustered in two groups as well, $\sim$800 and $\sim$1300. Finally, in Figure 4.3 (c) and (f), notice that the power follows the same distribution as the current.

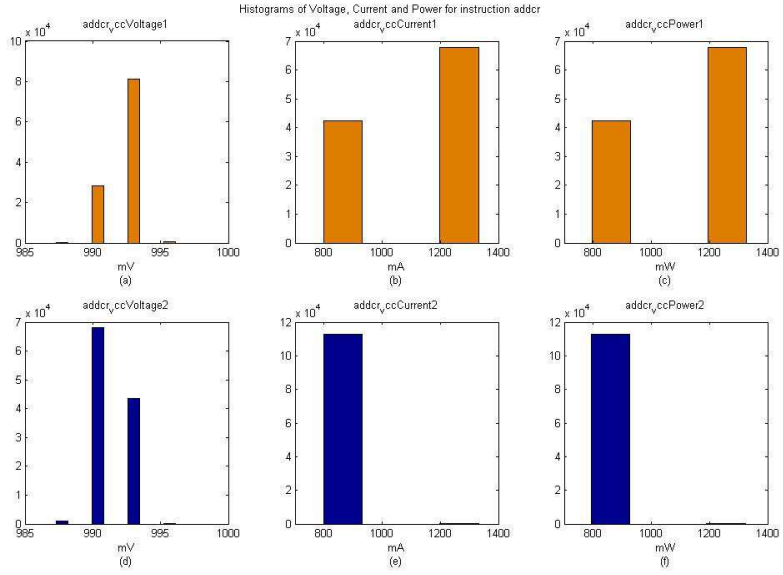


Figure 4.3: Histograms of Voltage, Current and Power measured for the addcr instruction

Now that we have confirmed that the current distribution defines the distribution of power consumption, we for an hypothesis on why we have only two groups in the current results:

As explained in section 3.2.1.1 the resolution of the current sensor is $1LSB_I = 443.892[mA]$. Then $2LSB_I = 887.784$ and $3LSB_I = 1331.676[mA]$ which are the groups identified in the histograms shown in Figure 4.3 (b) and (e). From the experimental results, we observe that the voltage drop in the Kelvin resistor is $2LSB_V = 1.954[mV]$ and $3LSB_V = 2.931[mV]$.

Validation of results with an Oscilloscope. As a result of the previous analysis and results, we assume that *the sensor used to indirectly measure current does not have the required resolution to measure the voltage drop caused by an AUT.*

In order to confirm the previous assumption, we performed physical measurements using an oscilloscope with a higher resolution than SysMon. This validation experiment is identified as PowerPC.Virtex5.GPP.A.1.4 throughout this thesis. With the oscilloscope, we measured the voltage dropped in the Kelvin resistor (the same one used by SysMon) to know the current in the power supply. [1].

The characteristics of the experiment PowerPC.Virtex5.GPP.A.1.4 are presented in Section 3.2.1. During this experiment, we observe the following:

1. The voltage drop in the Kelvin resistor is not a constant value, but a voltage signal with peaks and valleys. The voltage signal has two periodic peaks. Both peaks appear interleaved with a frequency of ∼590 KHz. The frequency of the bigger periodic peak (A) is ∼295 KHz which is the same as the small peak (B). Table 4.2 shows the detailed results of the measurements, but only the values of the bigger peak are presented.

2. The amplitude of one peak, in the voltage signal, is bigger than the other, and last for a longer period. The bigger peak (A) has a maximum amplitude of 65 mV for a duration of approximately 54-64 ns. The amplitude values in the valleys of the voltage signal are ∼1 mV. These results were obtained using different bit files with different AUTs.

In the Table 4.2 we present the results of the experiment PowerPC.Virtex5.GPP.A.1.4, that uses the oscilloscope to measure different AUT.

4.1.1.1 Conclusions

Based on the results obtained with the oscilloscope, and after a comparison with the results obtained with SysMon, we derive the following conclusions:

- The results measured with SysMon are an average of the voltage dropped in the kelvin resistor. As a result, if one voltage peak is detected, its contribution to the average voltage depends on the number of samples SysMon is able to obtain from the peak. In our experiments, the period while a voltage peak is different to zero is too small compared with the sampling frequency of SysMon, therefore even if a peak is detected only one sample will be obtained by SysMon and the remaining samples would be close to zero, therefore the effect of the peaks is small.

[1] The oscilloscope does not provide the functionality to transfer the measurement results to a PC, therefore we don not present graphs.

AUT	MaxPeak [mV]	Period of peak [ns]	Frequency(KHz)
Nothing running board	40	61	295
nop	62	60	295
addi	61	59	295
add	61	57	295
adde	61.6	56	295
and	61.6	57	295
lbz	61.6	59	295
machhw	61.6	59	295
CCU-addition	60.8	59	295
CCU-add-serial	64.8	57	295

Table 4.2: Results of physical measurements performed on a Virtex 5 using an oscilloscope

- Even if the voltage peaks are detected by SysMon, they might be overwritten by new measurements before we fetch them and store them, because the ADCs in SysMon can provide up to 200 Ksamples per second, and with our current *experimental framework* we can obtain around 540 samples per second. In the end, this also affects the non-repeatability observed in the preliminary results of SysMon.

- The number of samples obtained is principally limited by the performance of the Xilinx TCL interface used in the SysMon I/F module (presented in Figure 3.6).

The previous conclusions apply for measurements using SysMon in a Virtex 5 chip.

4.1.2 Experiment StrongARM.SimPanalyzer.GPP.B.1

This experiment uses a simulation framework to obtain the power consumed by a kernel, in this experiment we use the Sim-Panalyzer simulator. The experiment was designed to provide a higher flexibility of the architectural parameters of the processor which affect power consumption. We also design this experiment to validate our modelling methodology, since the experiment that uses physical measurements requires high-end measurement tools and a higher implementation time, compared to the simulation framework.

The main characteristics of this experiment are summarized in Table 4.3.

Parameter	Value
UUT	StrongARM GPP
Input of the model	Functions in C code
Measurement method	Simulation framework: Sim-Panalyzer 2.0.3
Clock Frequency	200 MHz
Power supply voltage	1.8

Table 4.3: Main characteristics of experiment StrongARM.SimPanalyzer.GPP.B.1

This experiment uses the Sim-Panalyzer, configured as a StrongARM processor, to obtain the power consumption of a kernel. The steps involved in this experiment were presented in detail in Chapter 3, and are presented here for reference:

- **Isolate the kernel**. This step is done once per each kernel.
- **Measure the power consumption**. This step is done using the data obtained from the ARGS tool. For each input dataset a measurement is done.
- **Obtain the SCMs**. This step is done once per each kernel.
- **Consolidate the power data**. After the measurements and all the metrics are obtained, we consolidate all the data in one single file.

After the previous steps are performed, we obtain a series of results that can be classified as follows:

- **Predictors or independent variables**. These are the values that are used to characterize the power consumption of a kernel. In our modelling process, these parameters are divided in two types: static and dynamic. Using this division, we present the parameters of each type, for this experiment:
  1. **Static**. The SCMs are the static parameters.
  2. **Dynamic**. The dynamic parameters are the *args.data metric* (obtained from ARGS tool), and the following parameters obtained from Sim-Panalyzer: the number of instructions commited (sim_num_insn), the number of loads and stores (sim_num_refs), the number of cycles (sim_cycle), and the cycles per instruction (sim_cpi).
- **Outcome variables or dependent variables**. These parameters are the object of our modelling process. We want to predict the energy and the power consumption of a kernel. As a result, in this experiment the following parameters, obtained from the Sim-Panalyzer, are used as dependent variables:
  1. **uarch.pdissipation**. This parameter is calculated in Sim-Panalyzer by adding the power consumed during each cycle, while an application is executed. It is important to notice that, this is not the energy of the application, it is instead, the accumulated power consumption of an application. In order to obtain the energy we use Equation 6.6, presented in Appendix 6.3, and multiply the *uarch.pdissipation* parameter with the clock period in this experiment (5 ns).
  2. **uarch.avgdissipation**. This parameter is the average power dissipation, consumed by an application, per cycle.

The results outlined above constitute the *Power Data*, shown in the modelling methodology of Figure 3.1.

As a summary of the *outcome variables*, we present in Figure 4.4 the energy consumed by each of the kernels in the *kernel library*. Only the identifier of each kernel is presented

in Figures 4.4 and 4.5, therefore in the Table 4.4 we present the identifiers associated with the name of the kernel.

It can be observed that the values of energy of the kernels differ from each other, with values in the range of $2.45348106 \times 10^{-6}$ up to 124.8455×10^{-6}. This variation depends mainly on the number of cycles the function takes to execute. A further analysis is presented later in this section, that provides a better explanation of the parameters that explain the energy consumed by a kernel.

In the Figure 4.5, we present the summary of the power consumption per kernel. It can be observed that in this case the power consumption is similar among the kernels, because the power consumed per kernel is an average of the power consumed per cycle. The values of power consumption are in the range of 380.481 mW up to 459.5 mW.

Since our model will be created using regression analysis in the *power data*, we performed first a preliminary analysis of the correlation between the dependent variables, and the independent variables selected for this experiment.

Firstly, we plotted the relation between energy and each one of the predictors, using a scatter plot. In the Figures 6.1, 6.2, 6.3, 6.4, 6.5, and 6.6 of Appendix 6.5, we present the scatter plots of the predictors with values different to zero. As a summary, in the Figure 4.6 we present the predictors with a strongest linear relation ship with the energy.

As part of the preliminary analysis, we plotted the relationship between power consumption and the predictors. As a summary, in the Figure 4.7, we present the predictors that have the strongest linear relation with the power consumed in this experiment. In Figures 6.7, 6.8, 6.9, 6.10, 6.11, and 6.12 in Appendix 6.6, we present the relation between power and all the non-zero predictors.

We can observe in Figure 4.6 that, the dynamic metrics have the strongest linear relation with the energy, among all the predictors. This can be explained with Equation 6.6,

ID	Kernel Name	ID	Kernel Name
0	g721_body	21	MD5Transform
1	mdct_butterfly_8	22	sha_transform
2	mdct_butterfly_16	23	apply_butterflies
3	CrossProduct3	24	serpent_encrypt
4	CrossProduct7	25	serpent_decrypt
5	bytesum	26	cast256_encrypt
6	polynomialbin	27	cast256_decrypt
7	intersect_triangle	28	gost_encrypt
8	intmatmult3x3	29	twofish_encrypt
9	intmatmult4x4	30	gost_decrypt
10	mseq	31	twofish_decrypt
11	Hamming1_unrolled	32	cast128_encrypt
12	Hamming2_unrolled	33	loki97_encrypt
13	hw_ripemd128_transform	34	loki97_decrypt
14	hw_ripemd160_transform	35	mrog_hw
15	hw_ripemd256_transform	36	complex_div
16	havalTransform3	37	complex_mult
17	MD4Transform	38	bitreversal2_unrolled
18	hw_ripemd320_transform	39	des_f
19	cast128_decrypt	40	max
20	base		

Table 4.4: Kernel IDs of the kernels used in experiment StrongARM.SimPanalyzer.GPP.B.1

that relates energy with power consumption.

$$E_{app} = \left(\sum^{num.cycles} P_{cycle} \right) \times T \tag{4.2}$$

However, this formula provides the energy consumed in one cycle, if we want to get the energy consumed by an application we need to sum the power consumed per cycle, and then, the accumulated power multiply it by the period of the processor's clock. The Equation 4.2 presents the formula used to calculate the energy in this experiment. As a result, the number of cycles and any metric related to the number of cycles, such as the number of instructions, or the number of loads and stores, will show a strong linear relation with the energy consumed in one application.

In Figure 4.7, we observe that also the dynamic parameters used as predictors have a strong linear relation with the power consumption of a kernel. However, opposite to the energy, the power consumption has an inverse linear relation with the dynamic parameters. This inverse linear relationship can be observed in almost all the other

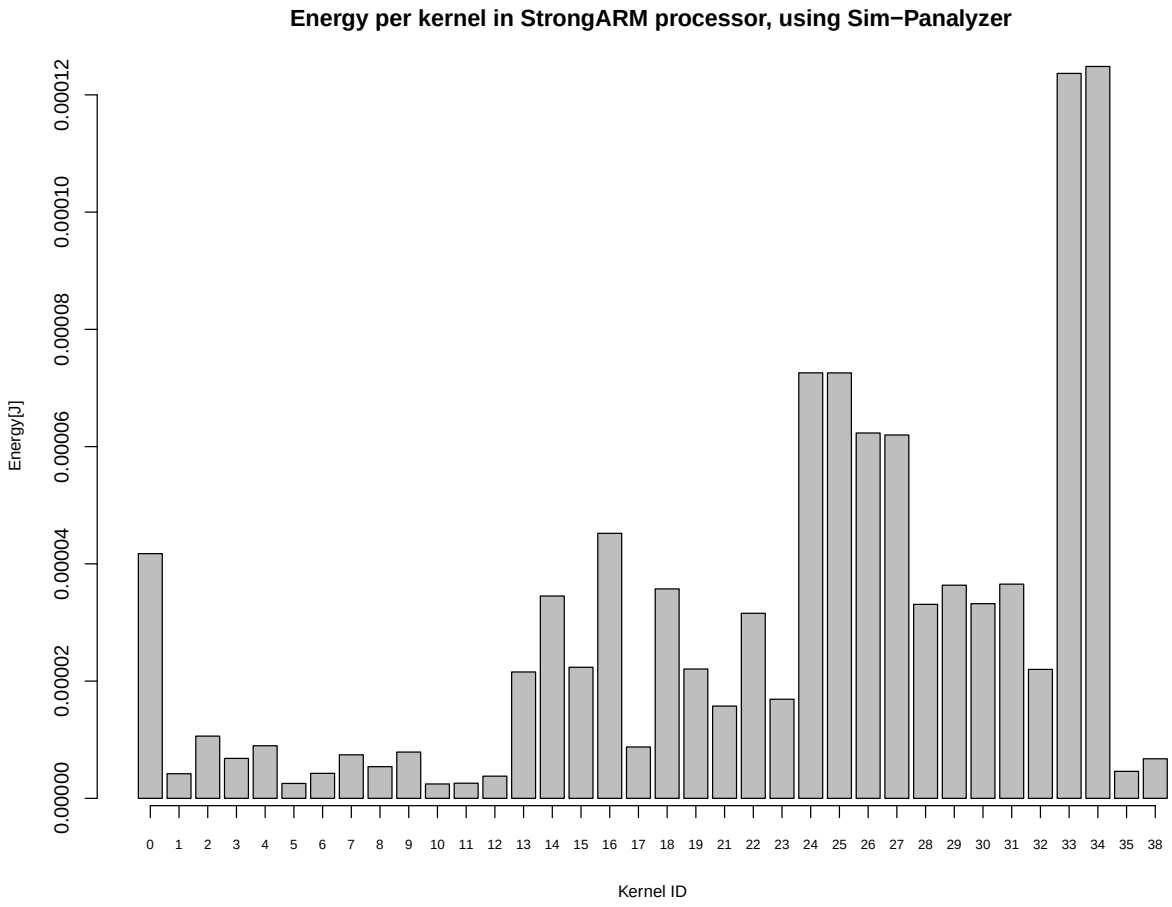


Figure 4.4: Experiment StrongARM.SimPanalyzer.GPP.B.1: Energy summary

predictors related to the number of cycles an kernel is executed, and could be explained by the *law of large numbers*. The *law of large numbers* defines that the average result of an experiment, will be closer to the expected value when more trials of the experiment are performed. In this experiment, the Sim-Panalyzer is measuring the power consumed per cycle, therefore in the experiments each trial will be the measurement of power in each cycle. For kernels with a high number of cycles, the average will tend to the expected value, and this would suggest that the power of the processor is higher at the startup stage and then would tend to a lower value. However, this assumption has to be verified with further experiments.

For both *outcome variables*, the energy and the power consumption, the dynamic parameters have the strongest linear relationship. However, they cannot be used in a model that uses as input HLL, because these parameters are known after executing the kernel in the Sim-Panalyzer simulator. Therefore, we have created two models, one model that uses only static parameters (SCMs), and another model that uses static and dynamic parameters. The results of the modelling stage are presented in the following

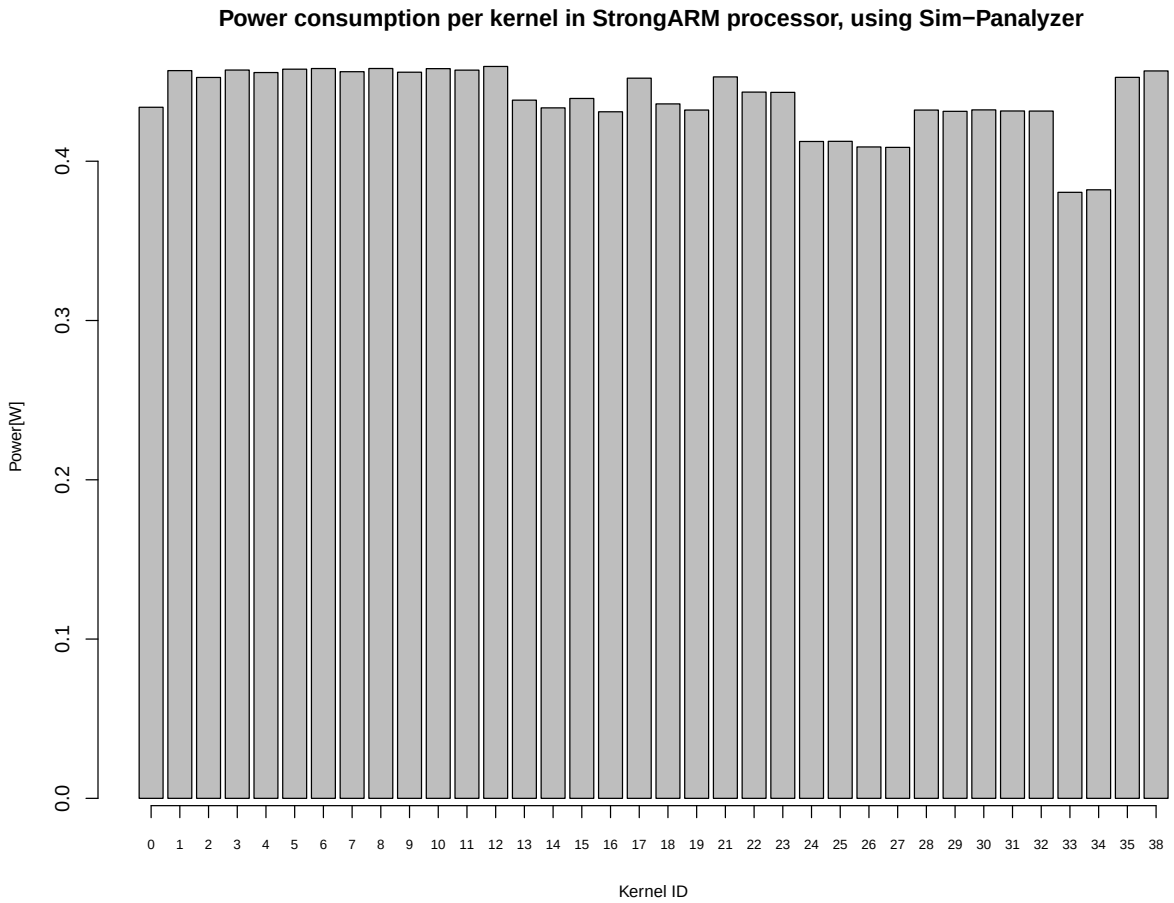


Figure 4.5: Experiment StrongARM.SimPanalyzer.GPP.B.1: Power summary

section.

4.1.2.1 Modelling results

In this section, we present the results of the modelling stage made with the Quipu approach [22].

For this experiment we have created four models, organized as follows:

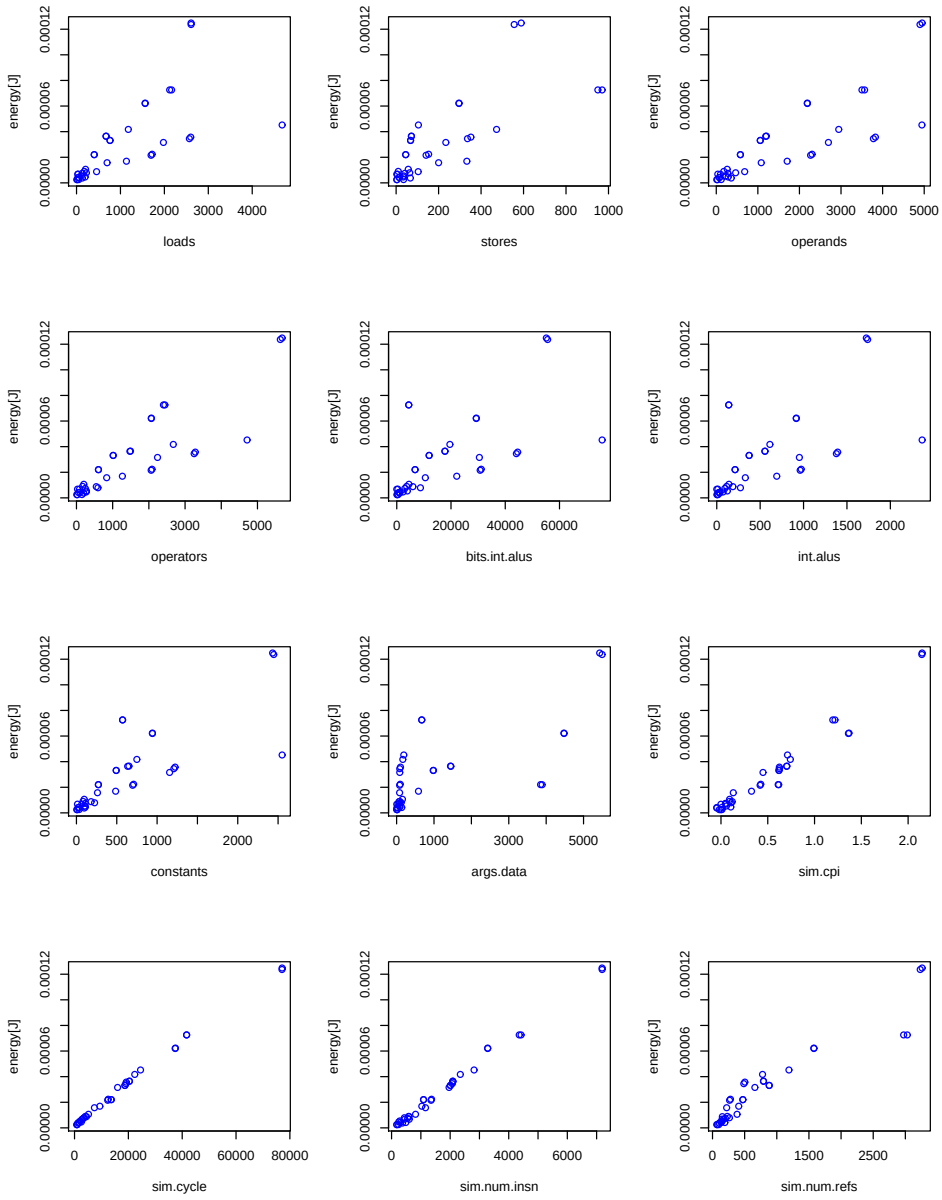


Figure 4.6: Relation between energy and its strongest predictors

- **Models created with static predictors**. One model was created for energy consumption, and another model for power consumption. Both models use the static predictors (SCMs) and require only the HLL (C-code) description of the kernel to predict the *outcome variables*.

- **Models created with static and dynamic predictors**. One model was created for energy consumption, and another model for power consumption. These models

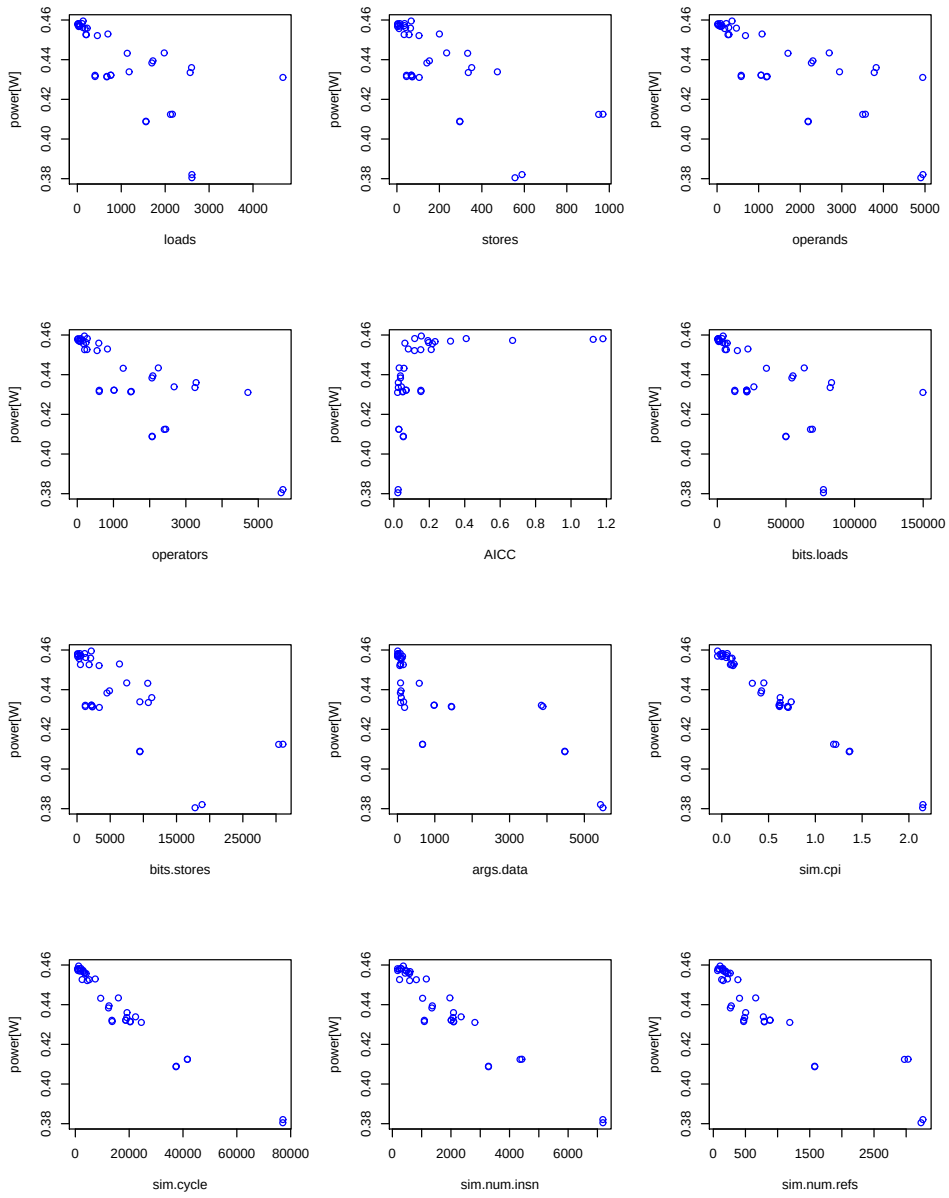


Figure 4.7: Relation between power and its strongest predictors

use the static predictors (SCMs) along with the dynamic predictors to predict the *outcome variables*. The accuracy of these models is better than the models that use only SCMs, however they require the execution of the kernel in the Sim-Panalyzer simulator, which increases the prediction time.

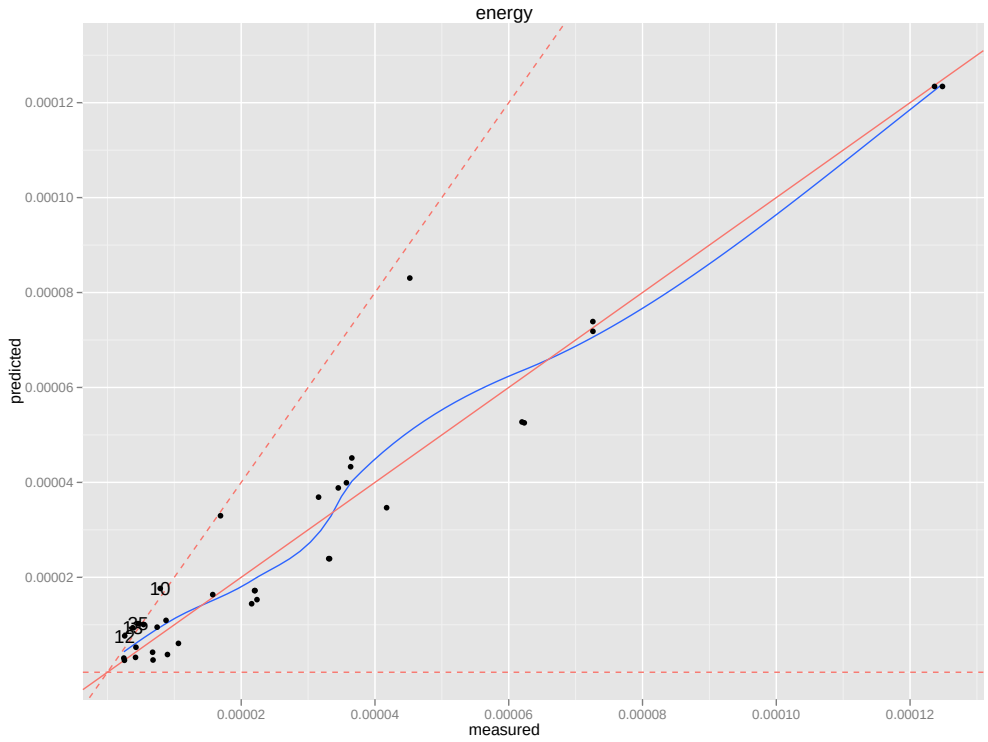


Figure 4.8: Energy model for the StrongARM GPP, with SCMs as predictors

Models using only static parameters(SCMs)

The Figure 4.8 presents the model that predicts the **energy consumption of a kernel, using only SCMs** as predictors. This figure shows the comparison between the measured values against the predicted values, the red line in the figure is the regression line of the model, and the blue line is the trend line of the actual data. The percentual relative rooted mean square error (RMSE) of the model is 29.69605%, and the absolute error of the model is 8.691191e-06 Joules. This model uses as input the HLL (C-code) description of a kernel, and the qpm-metricator tool [22], to perform the predictions of energy consumption of a kernel.

The Figure 4.9 presents the model that predicts the **power consumption of a kernel, using only SCMs** as predictors. This figure shows the comparison of measured values of power against predicted value. The red line in the figure is the regression line of the model, and the blue line is the trend line of the actual data. The percentual RMSE of the model is 1.943507%, and the absolute RMSE of the model is 8.526016 mW. This model only requires the SCMs and the qpm-metricator tool [22], to predict the power consumed by a kernel.

Models using static & dynamic parameters(SCMs)

The Figure 4.10 presents the model that predicts the **energy consumption of a kernel, using static and dynamic predictors**. The red line in the figure is the regression line of the model, and the blue line is the trend line of the actual data. The percentual relative rooted mean square error (RMSE) of the model is 3.949472%, and the absolute error of the model is 1.155898e-06 Joules. This model uses as input the HLL (C-code) description of a kernel, the qpm-metricator tool [22] to obtain the metrics, the ARGS tool to isolate the kernel, and the execution of the kernel in the Sim-Panalyzer to obtain the energy consumption of a kernel. Even though, the accuracy of this model is good, it cannot be used for prediction in this experiment, because you actually need to execute the kernel in the Sim-Panalyzer. This model would be useful in an experiment where running the kernel in the UUT is less complex than measuring the power(to obtain energy), such as the experiments with physical measurements.

The Figure 4.11 presents the model that predicts the **power consumption of a kernel, using static & dynamic predictors**. The red line in the figure is the regression line of the model, and the blue line is the trend line of the actual data. The percentual RMSE of the model is 0.754%, and the absolute RMSE of the model is 3.307784 mW. This model has the same disadvantage of the energy model, because it requires the actual execution of the kernel in the Sim-Panalyzer to obtain the dynamic metrics.

Models using static parameters, and only some selected dynamic metrics

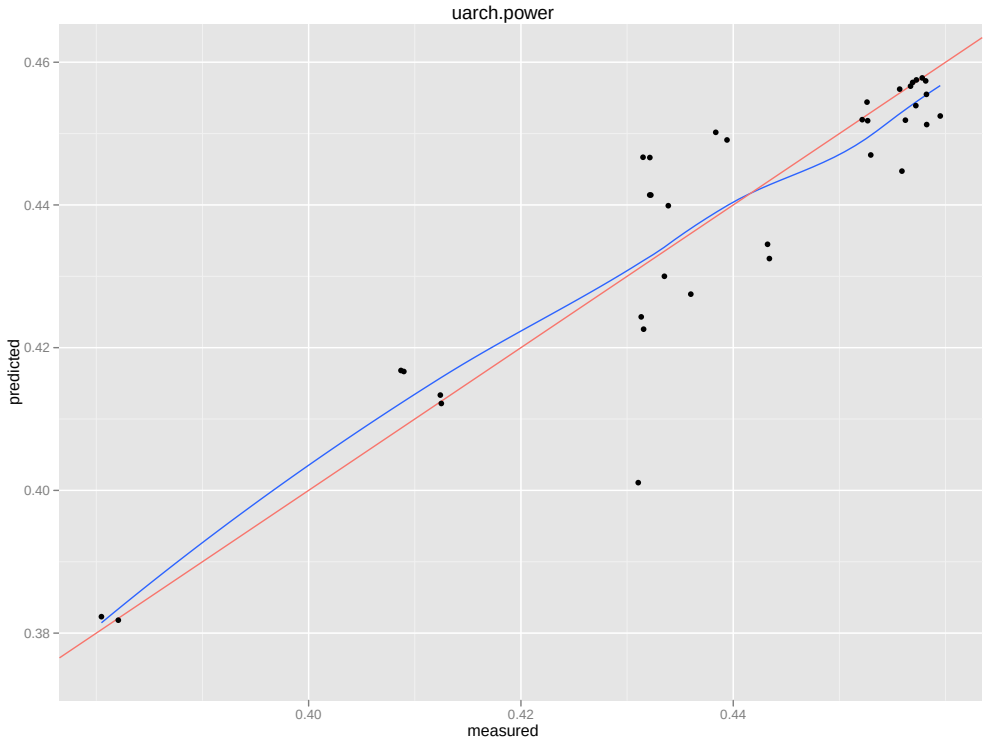


Figure 4.9: Power model for the StrongARM GPP, with SCMs as predictors

as predictors.

The models that use static metrics and all the dynamic metrics selected for this experiment have the disadvantage that cannot be used for prediction, because they require the actual execution of the kernel in the Sim-Panalyzer. However, if we use only static metrics and some specific dynamic metrics that do not require the exection of the kernel in the Sim-Panalyzer we can use the generated models for prediction.

As a proof of concept, we present in Figure 4.12 the energy model obtained using the SCMs as predictors but only selecting the number of instructions committed as dynamic predictor. We select the energy model, because it has a higher percentual RMSE compared with the power model, among all the models created in this experiment. The RMSE of this model is reduced, with a percentual RMSE of 21.87% and an absolute RMSE of 6.39942×10^{-6} J, for the energy model. The RMSE of this model is 7.8% smaller than the models that use only static parameters. In order to use this model for predictions, we assume that is easier to obtain the number of instructions committed by the StrongARM processor. As an example, we can use the SimpleScalar simulator, that does not report the power consumed by an application, but still can provide the number of instructions committed in the processor.

Other dynamic parameter that can be used to reduce the RMSE of the model that uses only static metrics, is the *args.data* metric. This parameter has the advantage that it is not required at all to execute the kernel in the UUT. We only need to execute the

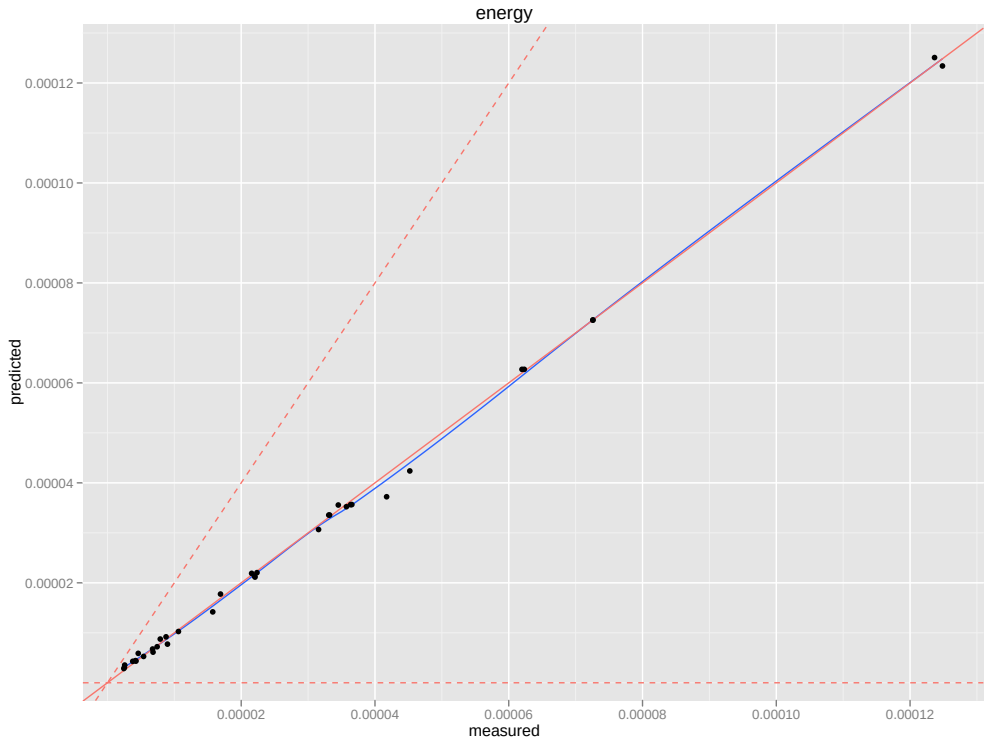


Figure 4.10: Energy model for the StrongARM GPP, with static & dynamic predictors

application in a x86 processor, using the ARGS tool, to obtain the *args.data metric*. The model of energy is presented in Figure 4.13. This model has an absolute RMSE of 8.166×10^{-6} J, and a percentual RMSE of 27.90%. Even though, only a reduction of 1.7% is obtained, this model can be used for predictions in the partitioning process because only requires SCMs and the execution of the application using the ARGS tool, in a x86 processor.

Model	Static		Static & dynamic		Static & num inst committed		Static & args.data	
	RMSEp(%)	Absolute RMSE	RMSEp(%)	Absolute RMSE	RMSEp(%)	Absolute RMSE	RMSEp(%)	Absolute RMSE
Energy	29.60695	8.691191×10^{-6} J	3.949472	1.15589810^{-6} J	21.87	6.39942×10^{-6}J	27.90	8.166×10^{-6}J
Power	1.943507	8.526016 mW	0.754	3.307784 mW	1.90	8.367244 mW	1.57	6.884843mW

Table 4.5: Comparison of the RMSE in the models created in experiment StrongARM.SimPanalyzer.GPP.B.1

In Table 4.5, we present a summary of the models created in this experiment. We can observe that the energy models present a higher percentual RMSE compared to the power models. The reason for this behavior can be explained as follows:

The kernels used during the modelling step have a range of energy values from 2.453481×10^{-6} J up to 124.8455×10^{-6} J, with a mean value of $2.926716\times10{-}5$. Using the model with only static predictors as example, we can observe that the absolute RMSE for the model is 8.691191×10^{-6} J, as a result, the absolute RMSE represents

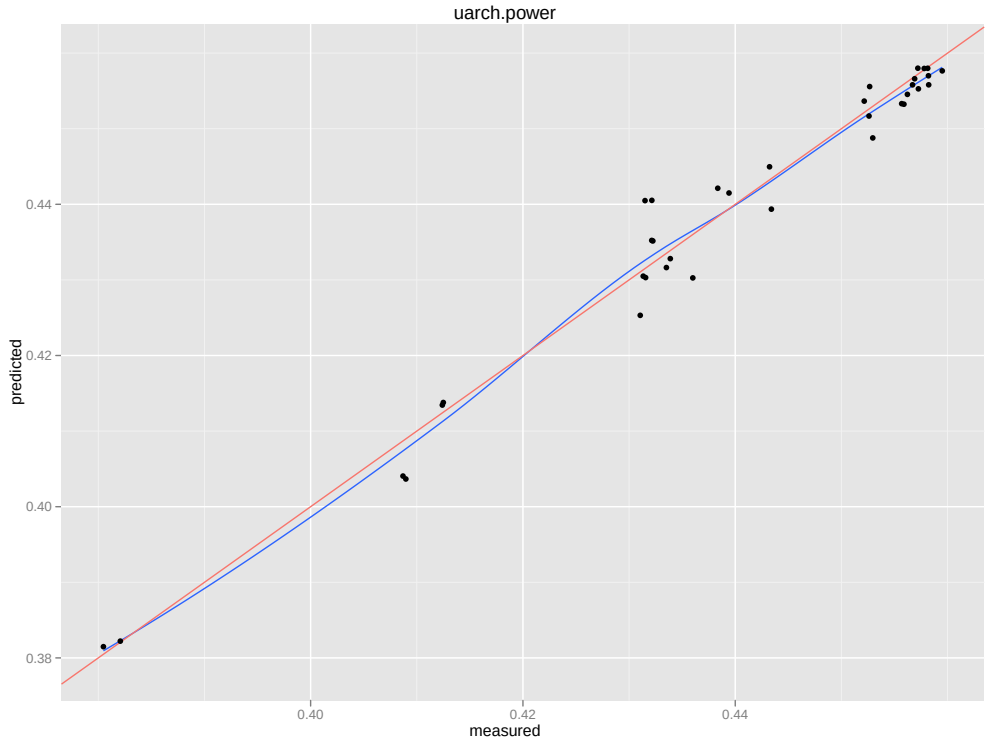


Figure 4.11: Power model for the StrongARM GPP, with static & dynamic predictors

around the 29% of the mean value. However for the power consumption, the kernels have a range of values from 380.481 mW up to 459.5 mW, with a mean value of 438.6924 mW. And the model with only static predictors have an absolute RMSE of 8.526016 mW that is approximately 2% of the mean value. However, we can observe that the minimum value of the power consumption is not zero, it is ∼380 mW, therefore the RMSE seems to be smaller in the power consumption model.

The previous analysis of power consumption suggest that a certain power consumption it is be used in the StronARM processor always, no matter which application is running. The power consumption we are referring to, is not only static power consumption but also dynamic power consumption (such as the power consumed by the clocking system). As a result, for a model that predicts power consumption it would be useful to isolate the threshold value of the power consumption consumed by the processor under any circumstance, and subtract it from the total power consumption of an application. This pre-processing step would be useful to provide a more realistic percentual error. Because of time constraints in this work, we did not calculate the threshold value of the power application. However, in this work we present also the absolute RMSE to provide a more realistic error of our power models.

In Table 4.5, can be observed that adding dynamic metrics to the modelling process reduces the RMSE of power and energy models. However, if we add specific dynamic

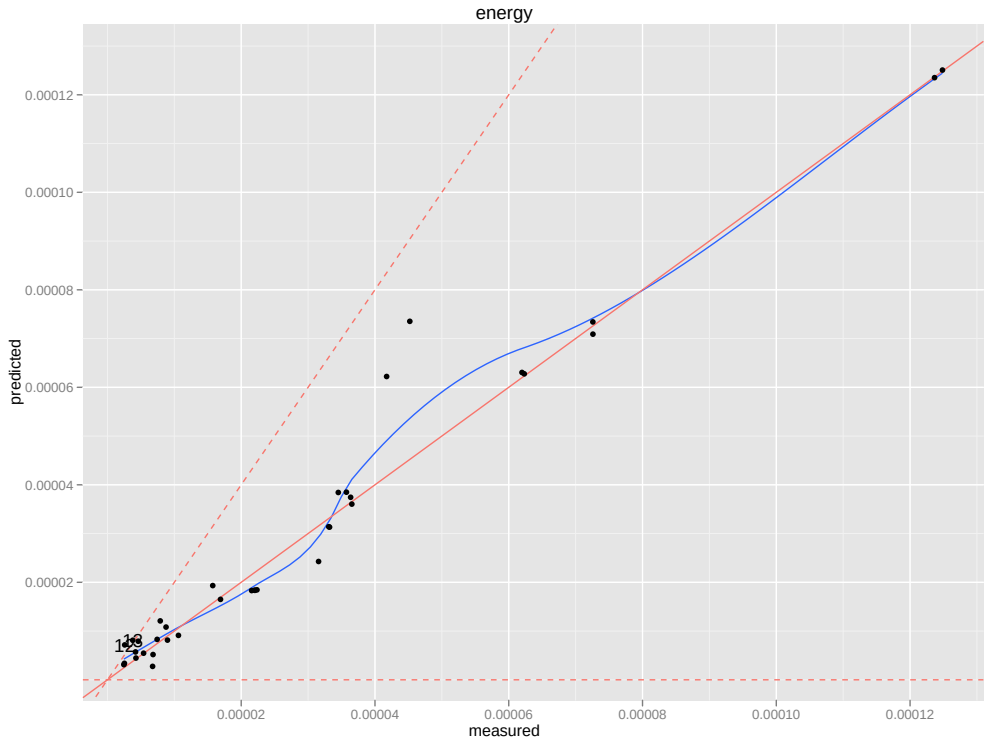


Figure 4.12: Energy model for the StrongARM GPP, with static predictors & the number of instructions commited as predictor.

metrics that do not require the execution of the kernel in the UUT, we can use the models, with a reduced RMSE, for prediction. We used the number of committed instructions, and the *args.data metric* as dynamic predictors. Each parameter was added separately in the modelling process, and based on the results we can observe that the number of instructions committed has a stronger relation with the energy compared with the *args.data metric*. The opposite is observed for the power consumption, the *args.data metric* has the strongest linear relationship among these two metrics. With the advantage that the *args.data metric* can be obtained by running the application of the kernel in a x86 processor, using the ARGS tool.

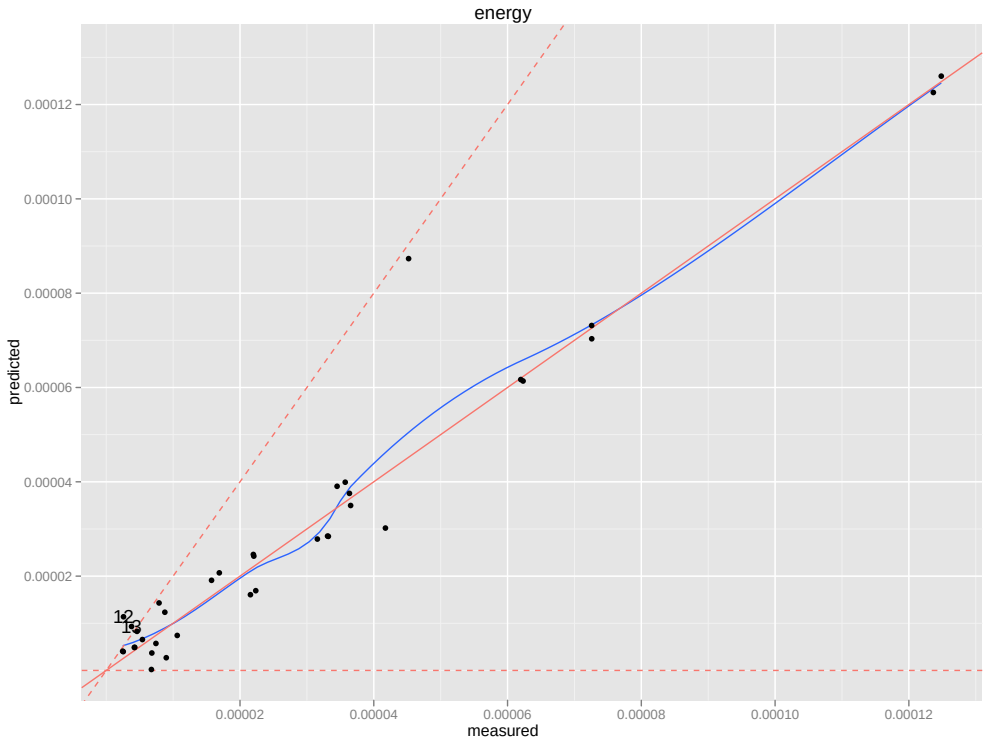


Figure 4.13: Energy model for the StrongARM GPP, with static predictors & the *args.data metric* as predictor.

4.1.3 Experiment VIRTEX5.Xpwr.FPGA.B.2

In this experiment we use the Virtex 5 FPGA as UUT, a different UUT compared with experiment StrongARM.SimPanalyzer.GPP.B.1. At the beggining of this section, we present a summary of the power data collected in this experiment. Then, after we performed the modelling step (using the Quipu approach) we obtain a set of models, of energy and power consumption, that validate the flexibility of the modelling methodology presented in this thesis.

The main characteristics of this experiment are summarized in the Table 4.6.

Parameter	Value
UUT	Virtex 5 FPGA XC5VFX130TFF1738-2
Input of the model	Functions in C code
Measurement method	Simulation framework: *xpwr* tool of Xilinx
Clock frequency	100 MHz
Power supply voltage	1v

Table 4.6: Main characteristics of the experiment VIRTEX5.Xpwr.FPGA.B.2

The *power data* obtained in this experiment can be classified as follows:

- **Predictors or independent variables**. We use this values to characterize the power and energy consumed by a kernel. The predictors used in this experiment were divided in static and dynamic, as follows:
  1. **Static**. The SCMs are the static predictors.
  2. **Dynamic**. The *args.data metric* was used as a dynamic predictor.
- **Outcome variables or dependent variables**. The following parameters are used as outcome variables:
  1. **Total power**. It is obtained with the *xpwr* tool of Xilinx.
  2. **Static power**. It is obtained with the *xpwr* tool of Xilinx.
  3. **Dynamic power**. It is obtained with the *xpwr* tool of Xilinx.
  4. **Energy**. It is calculated by multiplying the total power, the number of cycles, and the period of the clock used in this experiment.

As a summary of the *power data*, we present the Table 4.7 with the information of the total power, static power, dynamic power and energy consumed per kernel. We also present in the table, the figure number of each parameter.

In the Table 4.8, we present the kernel identifier associated with each kernel used in this experiment.

We can observe in the summary of the *power data*, that the static power consumption accounts for 92.5% of the power consumed by a kernel. This in general is one of the

	Min	Max	Mean	Percentual mean	Figure
Total power [mW]	2285.46	2838.15	2467.141	100	4.14
Static power [mW]	12.21	541.87	186.3447	7.5	4.16
Dynamic power [mW]	2273.25	2296.27	2280.797	92.5	4.15
Energy [mJ]	0.415	41.503	11.086	100	4.17

Table 4.7: Summary of the *power data* in experiment VIRTEX5.Xpwr.FPGA.B.2

ID	Kernel Name	ID	Kernel Name
1	mdct_butterfly_8	23	apply_butterflies
2	mdct_butterfly_16	26	cast256_encrypt
3	CrossProduct3	27	cast256_decrypt
4	CrossProduct7	28	gost_encrypt
5	bytesum	29	twofish_encrypt
6	polynomialbin	30	gost_decrypt
7	intersect_triangle	31	twofish_decrypt
8	intmatmult3x3	32	cast128_encrypt
9	intmatmult4x4	35	mrog_hw
10	mseq	36	complex_div
11	Hamming1_unrolled	37	complex_mult
12	Hamming2_unrolled	38	bitreversal2_unrolled
19	cast128_decrypt	39	des_f
22	sha_transform	40	max

Table 4.8: Kernel IDs of the kernels used in experiment VIRTEX5.Xpwr.FPGA.B.2

drawbacks of the FPGA based design, and can be explained by the way a design is implemented in an FPGA:

The FPGAs use look up tables (LUTs) to implement any logic in the reconfigurable fabrics. Each LUT stores all the possible input combinations in a design that can produce a result, moreover, if the functionality to be implemented requires more inputs than the available inputs in the LUT (6 for the Virtex 5 FPGA), an arrangement of LUTs is created to perform the required functionality. However, during the operation of the design not all the input combinations are valid. As a result, there is a lot of transistors that are not being used and consume static power. Only the transistors that switch during the operation of the design will account for the dynamic power consumption, that in this experiment is 7.5% of the total power consumption.

The energy consumed per kernel, summarized in the Figure 4.17, depends mainly on the number of cycles required by each kernel to perform its functionality.

Since the models in this experiment are generated using regression analysis, we present a preliminary analysis of the relation between the outcome variables and the predictors in the Table 4.9 using the linear correlation coefficient R. This coefficient express the strenght and direction of a linear relation between two variables, an outcome variable and a predictor. The values of the coefficient R are in the range of -1 to 1. The

value of the R coefficient express a strong positive linear relationship if it is closer to 1, and a strong negative relationship if it is closer to -1.

We can observe in the Table 4.9 that the following SCMs have a strong positive linear relationship with the power and energy consumed in the FPGA:

1. Number of load and stores in memory, and the related SCMs.
2. Number of constants, and the related SCMs.
3. Number of operands, and the related SCMs.
4. Number of operators, and the related SCMs.
5. Number of integer operations, and the related SCMs.
6. Number of statements, and the related SCMs.

An hypothesis for this strong relation, is that some of the outlined SCMs are related with the hardware resources that will be used for the kernel design in the FPGA. As an example, we have the number of operators and operands in a kernel, or the number of integer operations which will be translated into hardware resources during the C-to-VHDL translation. The SCMs related to loads and stores in memory, can be explained by the fact that the more read and write operations, in memory, the higher the energy

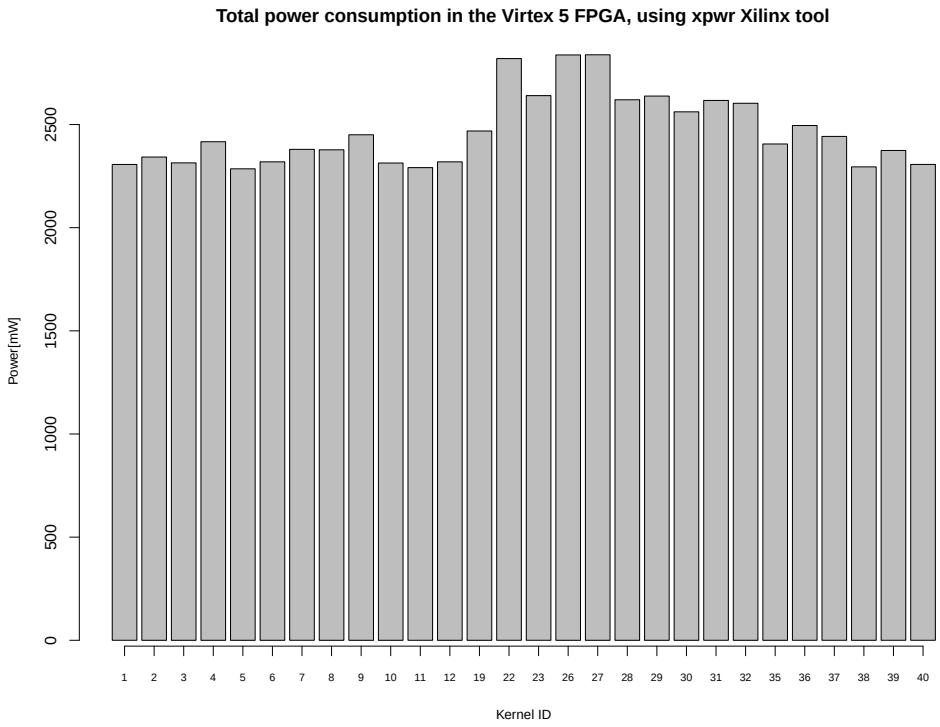


Figure 4.14: Experiment VIRTEX5.Xpwr.FPGA.B.2: Total power summary

will be. Also the power will be affected, because more switching activity is expected in the routing elements that transport the data from/to memory, and in the transistors that implement the memory, if more R/W operations are performed.

4.1.3.1 Modelling results

In this experiment we have created 8 models that predict the energy, the total power, the static power, and the dynamic power consumed by a kernel implemented in the Virtex 5 FPGA.

The models are organized as follows:

- **Using only static predictors**. This models use a set of SCMs to predict the following outcome variables of a kernel:
  1. **Energy**
  2. **Total power**
  3. **Static power**
  4. **Dynamic power**
- **Using static & dynamic predictors**. This models use a set of SCMs and the *args.data metric* to predict the following outcome variables of a design:

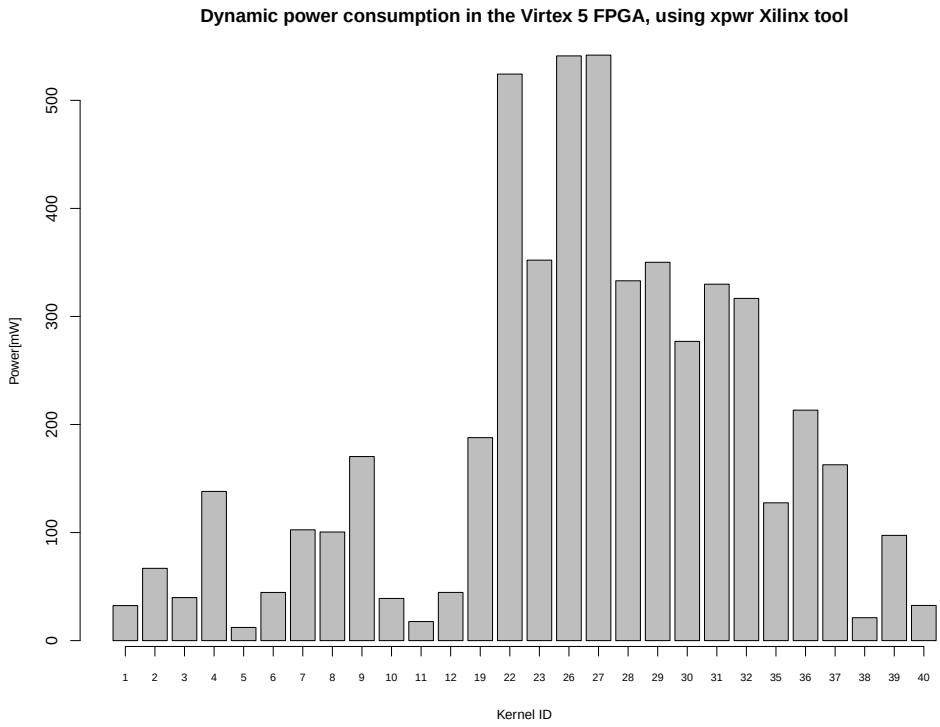


Figure 4.15: Experiment VIRTEX5.Xpwr.FPGA.B.2: Dynamic power summary

1. **Energy**
2. **Total power**
3. **Static power**
4. **Dynamic power**

In Table 4.10, we present a summary of the RMSE of the models created with the Quipu modelling approach. Figures 6.13, 6.14, 6.15, 6.16, 6.17, 6.18, 6.20, and 6.19 in Appendix 6.7, present the models created in this experiment.

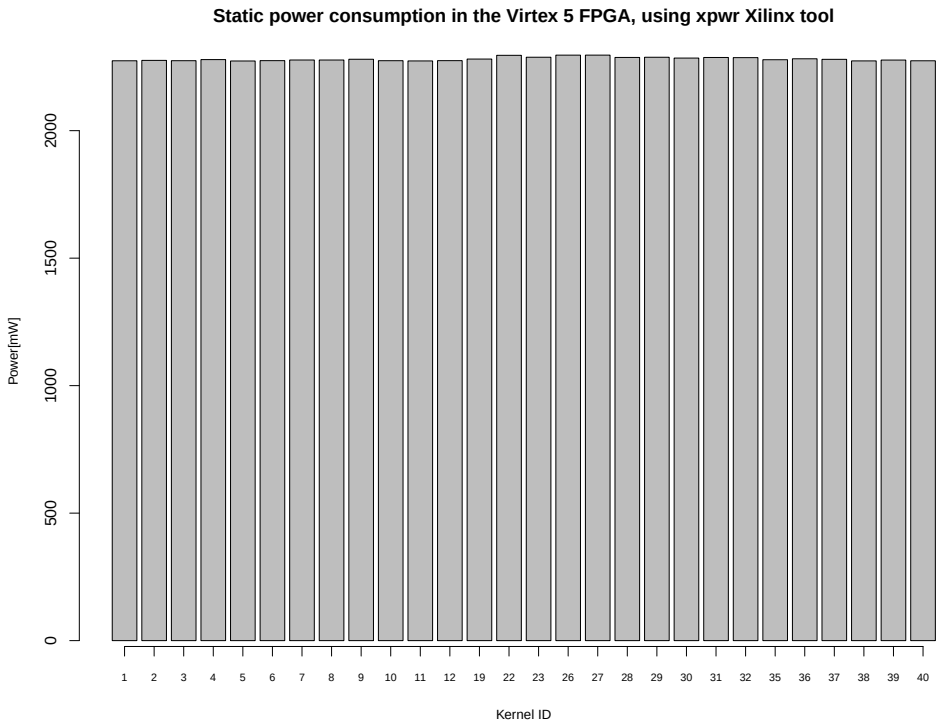


Figure 4.16: Experiment VIRTEX5.Xpwr.FPGA.B.2: Static power summary

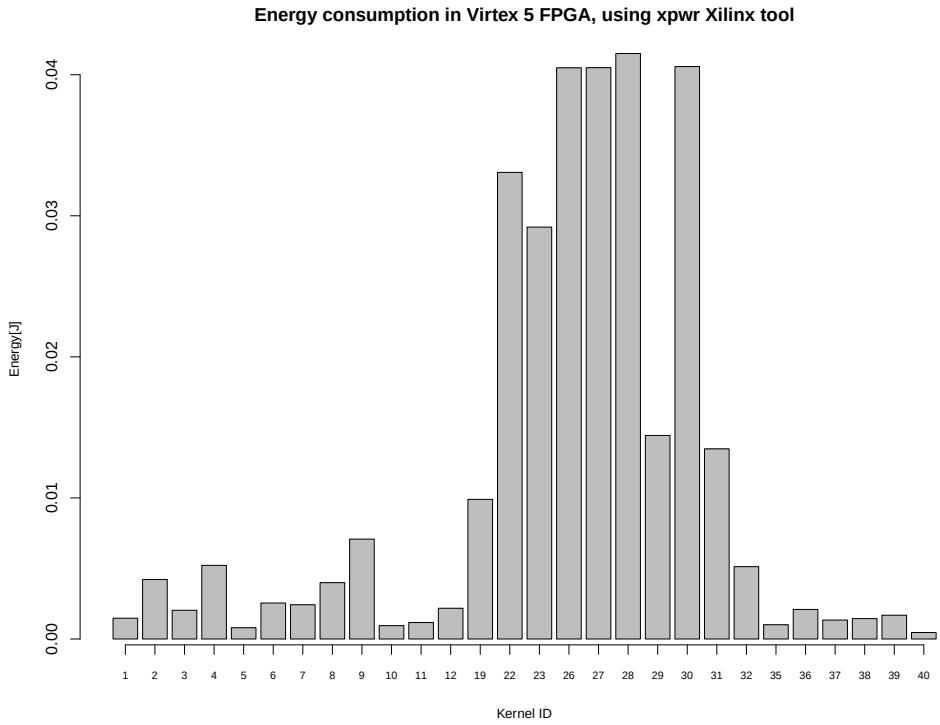


Figure 4.17: Experiment VIRTEX5.Xpwr.FPGA.B.2: Energy summary

	energy	ccu.total.pdissipation	ccu.static.dissipation	ccu.dyn.dissipation
heap.loads	0.89989485	0.931649977	0.931627275	0.93217827
loads	0.88335151	0.929058046	0.929038384	0.929546919
operands	0.87259857	0.932552451	0.932535114	0.932981802
bits.loads	0.8637448	0.929043472	0.929022766	0.929559195
bits.constants	0.85919732	0.938778354	0.938763244	0.939147614
constants	0.85913443	0.938709416	0.938694287	0.93907906
bb.max.heap.loads	0.85802025	0.890072765	0.890046511	0.890703286
bits.int.alus	0.85672267	0.939572551	0.939556729	0.939923022
int.alus	0.85672267	0.939572551	0.939556729	0.939923022
executed.operators	0.85604941	0.945221111	0.945208018	0.945519238
operators	0.85604941	0.945221111	0.945208018	0.945519238
bits.heap.loads	0.83637441	0.922196557	0.92216944	0.922829769
bb.max.loads	0.83054811	0.882829047	0.882804615	0.883443207
int.constantshifts	0.80190314	0.753718915	0.753713011	0.75397901
bb.max.operators	0.79930949	0.891164958	0.891148455	0.891558041
bits.stores	0.77928359	0.814324892	0.814300495	0.81486564
stores	0.77554197	0.809015315	0.808989909	0.809573296
bb.avg.heap.loads	0.76834286	0.777315285	0.7773057	0.777680971
bb.max.statements	0.75886868	0.828184559	0.828146638	0.829031196
bb.avg.loads	0.71680182	0.752075166	0.752067348	0.752418055
executed.statements	0.71175022	0.729818652	0.729798767	0.730248494
statements	0.71175022	0.729818652	0.729798767	0.730248494
bb.avg.statements	0.70434341	0.774030036	0.774008934	0.774635524
bits.bitwise	0.69076686	0.72190295	0.721893374	0.722319237
bitwise	0.69076686	0.72190295	0.721893374	0.722319237
bb.avg.operators	0.67007311	0.75064251	0.750644255	0.750717145
uOperands	0.60218101	0.685698304	0.685689812	0.685732682
bits.heap.stores	0.55108243	0.57572993	0.575713768	0.576113645
heap.stores	0.54912461	0.579600509	0.579584686	0.579970684
args.data	0.53141272	0.671079164	0.671069571	0.671118453
int.barrelshifts	0.52738113	0.621368937	0.621331955	0.622039084
type.conversions	0.51099624	0.653419621	0.653422567	0.653179582
oviedo	0.4990405	0.636551055	0.636564159	0.636214366
bits.type.conversions	0.49092611	0.599742425	0.599754134	0.599313504
bits.int.constmults	0.46402466	0.684160686	0.684145369	0.68434332
int.constmults	0.46402466	0.684160686	0.684145369	0.68434332
uOperators	0.42316937	0.561539984	0.561544694	0.561286023
scope.number	0.23337379	0.186511571	0.186523505	0.186208056
locals	0.22352299	0.204887777	0.204903749	0.204476191
pointers	0.22261314	0.273312631	0.273345095	0.272378969
bits.int.comps	0.20087736	0.161258349	0.16126926	0.160913425
int.comps	0.20087736	0.161258349	0.16126926	0.160913425
basicblocks	0.1707508	0.119877736	0.119886504	0.119654282
parameters	0.15933707	0.191402402	0.19142759	0.190630754
cyclomatic	0.14468324	0.085806282	0.085813374	0.085683276
bits.parameters	0.13715151	0.175305129	0.175334366	0.174459615
bits.locals	0.09168063	0.123880668	0.123907021	0.123209412
scope.ratio	-0.02181817	-0.039110065	-0.039095985	-0.039281809
bits.int.tests	-0.11536864	-0.160028383	-0.16003587	-0.159622483
int.tests	-0.11536864	-0.160028383	-0.16003587	-0.159622483
bits.fp.divs	-0.12003402	0.031702118	0.031722311	0.031230274
fp.divs	-0.12003402	0.031702118	0.031722311	0.031230274
bits.fp.mults	-0.16804461	0.016168004	0.016197098	0.015609934
fp.mults	-0.16804461	0.016168004	0.016197098	0.015609934
bits.fp.alus	-0.17620488	-0.007646043	-0.007614817	-0.008141846
fp.alus	-0.17620488	-0.007646043	-0.007614817	-0.008141846
bits.int.mults	-0.1773283	-0.14542688	-0.145412051	-0.145893794
int.mults	-0.1773283	-0.14542688	-0.145412051	-0.145893794
AICC	-0.49050842	-0.475722426	-0.475727236	-0.475482304
elshof.data.flow	-0.51409645	-0.462686561	-0.462681508	-0.462718771
returns.value	-0.55625729	-0.670256778	-0.670284566	-0.669657585

Table 4.9: Correlation coefficients of predictors in the experiment VIRTEX5.Xpwr.FPGA.B.2

	Static		Static & dynamic	
Model	RMSEp(%)	Absolute RMSE	RMSEp(%)	Absolute RMSE
Energy	14.34034	1.5898610^{-6} J	44.61419	4.9462110^{-6} J
Total power	2.4	60.13573 mW	1.318313	32.52463 mW
Static power	1.092485	2.491736 mW	1.091853	2.490294 mW
Dynamic power	30.93404	57.64394 mW	18.3595	34.21195 mW

Table 4.10: Summary of the models created in the experiment VIRTEX5.Xpwr.FPGA.B.2

4.2 Validation

The models presented in this thesis were automatically validated using the Quipu approach. Quipu uses for validation of the models, a method called *K-fold cross-validation*. In the K-fold cross-validation the data set is divided in two subsets, the training set and the validation set. The training subset is used to bootstrap the model using regression analysis. The validation set is not included in the regression analysis, and instead is used to validate the model. During the cross-validation process, the training and validation subsets are changed, and the error reported at the end of the process is the average error among all the different predictions performed with the different subsets.

Depending on the dataset Quipu performs the cross-validation with different values of k to obtain the lowest RMSE. For the models generated in this thesis, the value selected by Quipu is k=5.

Conclusions and future research.

5

In this thesis, we proposed a high level modelling methodology to create power and energy prediction models using the Quipu approach.

5.1 Summary

In Chapter 2, we presented the contextual framework of this work. We introduced the background and related research, in the context of power and energy prediction for the partitioning process of the DWB design flow. In Chapter 3, we proposed a modelling methodology that generates prediction models to provide quantitative data to guide the partitioning process. The prediction models require high level metrics, the SCMs proposed in [22], to predict the power and energy consumed by a function in different processing elements of heterogeneous architectures. The main characteristic of the modelling methodology is the separation of the modelling process in two steps:

1. The isolation of the dynamic behavior of an application.
2. The measurement of the power consumed by the application.

The separation of the modelling process provides the *flexibility* to investigate different architectural parameters that characterize the power and energy consumed in different processing elements. This separation also provides the *scalability* that allows the generation of models for different processing elements of heterogeneous architectures. Using this methodology, we generated a *power dataset* that is analyzed using different regression techniques, performed by the Quipu modelling approach, to generate prediction models of power and energy consumption. In Chapter 4, we presented the results for the validation of the proposed modelling methodology. For this purpose, we generated of a set of models for a StrongARM GPP and a Virtex 5 FPGA. The characteristics of these models, in terms of the RMSE, are different for energy and power. We present in Table 5.1, the RMSE of the models, which require only the SCMs of a C function to predict the outcome variable of the model. In Table 5.2, we present the RMSE of the models using SCMs and some dynamic metrics as predictors.

Model	Static		Static & dynamic	
	RMSEp(%)	Absolute RMSE	RMSEp(%)	Absolute RMSE
Energy	29.60695	8.69119110^{-6} J	3.949472	1.15589810^{-6} J
Power	1.943507	8.526016 mW	0.754	3.307784 mW

Table 5.1: Summary of the RMSE of the models using SCMs as predictors

	Static & num inst committed		Static & args.data	
Model	RMSEp(%)	Absolute RMSE	RMSEp(%)	Absolute RMSE
Energy	21.87	6.39942×10^{-6}J	27.90	8.166×10^{-6}J
Power	1.90	8.367244 mW	1.57	6.884843mW

Table 5.2: Summary of the RMSE of the models using SCMs & some dynamic metrics as predictors

Based on the research work conducted in this thesis, we have derived the conclusions presented in Section 5.2. In Section 5.3, we highlight some future work directions.

5.2 Conclusions

The conclusions of this research work are summarized as follows:

- The proposed modelling methodology provides a structured and automatic approach that can generate prediction models of energy and power consumption for different processing elements of a heterogeneous architecture. Considering the processing element as a black-box in this methodology provides the scalability that allow an easy change of processing elements.

- The SCMs, used for characterization of power and energy, provide an adequate accuracy to predict energy and power consumption. As an example, with a RMSE of 29.6%, using only 28 kernels in the modelling process of the StrongARM processor, we were able to create an energy model that provides quantitative data to guide the partioning process of the HW/SW co-design. If the number of kernels is increased, this methodology would exhibit a better accuracy. As an example, one of our experiments using 41 kernels, we generate an energy model for the Virtex 5 FPGA with a RMSE of 14.3%.

- Even though the physical measurements on a chip would provide more accurate values for power consumption, which results in better models of power and energy, using a simulation framework provides a higher flexibility. This flexibility enable us to explore different architectural parameters of a processing element, which can help in characterizing the power and energy consumed by a function. We believe that our modelling approach takes more relevance with the high number of different processing elements available in the market nowadays, because the exploration of main architectural parameters by using physical measurements on a chip would make the modelling process completely a complex task.

- The power and energy consumed by a function in different processing elements, is highly dependent on the characteristics of the underlying hardware where the function is running. Therefore, the separation of the dynamic behavior of an application and its power consumption, during the modelling process of power and energy, is a relevant step to cope with the rapid development of processing elements. This separation can provide an easy way to abstract away the effects of the architectural parameters of a processing element and reduce the time used to create the prediction models.

- Considering the increasing efforts of hardware manufacturers to reduce the static power consumption of processing elements, the significance of the proposed methodology is exposed in providing an structured and automatic approach to predict dynamic power consumption.

5.3 Future research

In order to improve the work presented in this thesis, we suggest the following:

- Including more kernels in the modelling process is required to improve the accuracy and the applicability of the proposed approach.
- The automatic modelling approach, proposed in this thesis, allows the exploration of different architectural parameters that can characterize the power and energy consumption of a kernel. As an inmediate proposal, we suggest to include different architectural parameters in the modelling process to improve the applicability of this modelling approach to different architectures.
- In the scope of this thesis, we created power and energy prediction models for kernels without loops, however with the ability to extract real data passed to a fuction, through the arguments, we can relate this information with the SCMs that can describe the function in terms of loops. Therefore, we suggest the inclusion of kernels with loops as a first step to improve the applicability of this modelling approach.
- With the proposed modelling methodology it is possible to model different processing elements, in consequence we suggest the modelling of processing elements, such as, a DSP or a GPU. This would increase the applicability of the proposed modelling approach.

Glossary

$\rho\mu$-code	Reconfigurable micro-code, 23
ADC	Analog to Digital Converter, 47
ALU	Arithmetic Logic Unit, 12
ANN	Artificial Neural Networks, 27
API	Application Programming Interface, 25
ASIC	Application Specific Integrated Circuit, 7
AUT	Action Under Test, 46
CAD	Computer Aided Design, 8
CCU	Custom Computing Unit, 50
CCU	Custom Configured Unit of MOLEN machine organization, 24
CFG	Control Flow Graph, 20
CLB	Configuration Logic Block, 10
DBA	Dynamic Binary Analysis, 25
DSE	Design Space Exploration, 16
DSP	Digital Signal Processor, 7
DWB	Delft Workbench, 7
EA	Effective Address, 38
EBP	Extended Base Pointer, 38
ELF	Executable and Linkable Format, file, 49
ESP	Extended Stack Pointer, 37
FIFO	First In First Out, 49
FIR	Finite Impulse Response, 10
FLPA	Functional Level Power Analysis, 12
FPGA	Field Programmable Gate Array, 7
GLM	Generalized Linear Model, 28
GPP	General Purpose Processor, 7
GUI	Graphical User Interface, 49
HDL	Hardware Description Language, 10
ILPA	Instruction Level Power Analysis, 12
IP-Cores	Intellectual Property Cores, 12
ISA	Instruction Set Architecture, 14

JIT	Just In Time, 40
JTAG	Joint Test Action Group, interface, 48
LIFO	Last In First Out, 38
LRM	Linear Regression Model, 28
LSB	least significant bit, 48
LUT	Look Up Table, 78
LUT	Look-Up Table, 8
Macro-model	Model with a coarse (low) level of detail [15]., 10
MAF	Memory Access Filter: is a module in ARGS, 41
MAT	Memory Access Tracing, 25, 42
NOP	No Operation assembly instruction, 18
PLB	Processor Local Bus, 50
PLSR	Partial Least Squares Regression, 28
PMDS	Permanent Memory Data Storage: a module in Pin, 43
PMT	Power Measurement Tool, 49
QUAD	Quantitative Usage Analysis of Data, 25
RMDS	Runtime Memory Data Storage: is a module in ARGS, 41
RMS	Root Mean Square, 13
RMSE	Rooted Mean Square Error, 72, 73
RMSEp	Relative Mean Square Error of prediction, used in K-fold cross-validation, 29
RP	Reconfigurable Processor, 23
RTL	Register Transfer Level, 7
SCM	Software Complexity Metrics, 27, 34, 44
SDL	System design level, 1
SMS	Stepwise Model Selection, 28
SysMon	System Monitor, an on-chip HW module in the Virtex 5 chip, 47
TTM	Time to market, 1
UUT	Unit Under Test, 45

VM	Virtual Machine, 25
XREG	Exchange registers of MOLEN machine organization, 23

Bibliography

[1] Syed Saif Abrar, *Cycle-accurate energy model and source-independent characterization methodology for embedded processors*, Proceedings of the 17th International Conference on VLSI Design (Washington, DC, USA), VLSID '04, IEEE Computer Society, 2004, pp. 749–.

[2] O. Acevedo-Patinando, M. Jimeandnez, and A.J. Cruz-Ayoroa, *Static simulation: A method for power and energy estimation in embedded microprocessors*, Circuits and Systems (MWSCAS), 2010 53rd IEEE International Midwest Symposium on, 2010, pp. 41 –44.

[3] Y. Ben-Asher and N. Rotem, *Synthesis for variable pipelined function units*, System-on-Chip, 2008. SOC 2008. International Symposium on, nov. 2008, pp. 1 –4.

[4] Yosi Ben-Asher and Nadav Rotem, *Automatic memory partitioning: increasing memory parallelism via data structure partitioning*, Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (New York, NY, USA), CODES/ISSS '10, ACM, 2010, pp. 155–162.

[5] Koen Bertels, Stamatis Vassiliadis, Elena Moscu Panainte, Yana Yankova, Carlo Galuzzi, Ricardo Chaves, and Georgi Kuzmanov, *Developing applications for polymorphic processors: The delft workbench.*

[6] Bluebee, *Bluebee white paper*, `http://www.bluebee-tech.com/downloads/BlueBee_Whitepaper.pdf`, 2011, Website.

[7] Box, G. E. P. and Cox, D. R., *An analysis of transformations*, Journal of the Royal Statistical Society. Series B (Methodological) **26** (1964), no. 2, 211–252.

[8] S. Chandrasekaran and A. Amira, *Power reduction for fpga implementations : Design optimisation and high level modelling*, Field Programmable Logic and Applications, 2006. FPL '06. International Conference on, August 2006, pp. 1 –2.

[9] ———, *A new behavioural power modelling approach for fpga based custom cores*, Proceedings of the Second NASA/ESA Conference on Adaptive Hardware and Systems (Washington, DC, USA), IEEE Computer Society, 2007, pp. 350–357.

[10] Naehyuck Chang, Kwanho Kim, and Hyung Gyu Lee, *Cycle-accurate energy measurement and characterization with a case study of the arm7tdmi*, IEEE Trans. Very Large Scale Integr. Syst. **10** (2002), 146–154.

[11] Vijay Degalahal and Tim Tuan, *Methodology for high level estimation of fpga power consumption*, Proceedings of the 2005 Asia and South Pacific Design Automation Conference (New York, NY, USA), ASP-DAC '05, ACM, 2005, pp. 657–660.

[12] David Elleouet, Yannig Savary, and Nathalie Julien, *An fpga power aware design flow*, Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation (Johan Vounckx, Nadine Azemard, and Philippe Maurine, eds.), Lecture Notes in Computer Science, vol. 4148, Springer Berlin / Heidelberg, 2006, pp. 415–424.

[13] Daniel D. Gajski, Samar Abdi, Andreas Gerstlauer, and Gunar Schirner, *Embedded system design: Modeling, synthesis and verification*, 1st ed., Springer Publishing Company, Incorporated, 2009.

[14] D. Grune, H. Bal, C. Jacobs, and K. Langendoen, *Modern Compiler Design*, Wiley, August 2000.

[15] Ali K. Gunal, Edward J. Williams, and Shigeru Sadakane, *Modeling of chain conveyors and their equipment interfaces*, Proceedings of the 28th conference on Winter simulation (Washington, DC, USA), WSC '96, IEEE Computer Society, 1996, pp. 1107–1114.

[16] P. Jamieson, W. Luk, S.J.E. Wilton, and G.A. Constantinides, *An energy and power consumption analysis of fpga routing architectures*, Field-Programmable Technology, 2009. FPT 2009. International Conference on, dec. 2009, pp. 324 –327.

[17] Tianyi Jiang, Xiaoyong Tang, and Prith Banerjee, *Macro-models for high level area and power estimation on fpgas*, Proceedings of the 14th ACM Great Lakes symposium on VLSI (New York, NY, USA), GLSVLSI '04, ACM, 2004, pp. 162–165.

[18] Hyung Gyu Lee, Kyungsoo Lee, Yongseok Choi, and Naehyuck Chang, *Cycle-accurate energy measurement and characterization of fpgas*, Analog Integr. Circuits Signal Process. **42** (2005), 239–251.

[19] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood, *Pin: building customized program analysis tools with dynamic instrumentation*, SIGPLAN Not. **40** (2005), 190–200.

[20] P. Marwedel, *Embedded system design*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[21] T.J. McCabe, *A complexity measure*, Software Engineering, IEEE Transactions on **SE-2** (1976), no. 4, 308 – 320.

[22] R. J. Meeuws, C. Galuzzi, and K.L.M. Bertels, *High level quantitative hardware prediction modeling using statistical methods*, Proceedings of the International Conference on Embedded Computer Systems: Architectures, Models, and Simulations, July 2011, pp. 140–149.

[23] R. J. Meeuws, K Sigdel, Y. D. Yankova, and K.L.M. Bertels, *High level quantitative interconnect estimation for early design space exploration*, ICFPT '08: Proceedings of the 2008 International Conference on Field-Programmable Technology, December 2008, p. 4.

[24] R. J. Meeuws, Y. D. Yankova, K.L.M. Bertels, G. N. Gaydadjiev, and S. Vassiliadis, *A quantitative prediction model for hardware/software partitioning*, Proceedings of 17th International Conference on Field Programmable Logic and Applications (FPL07), August 2007, pp. 735–739.

[25] H. Mehta, R. M. Owens, and M. J. Irwin, *Instruction level power profiling*, Proceedings of the Acoustics, Speech, and Signal Processing, 1996. on Conference Proceedings., 1996 IEEE International Conference - Volume 06 (Washington, DC, USA), ICASSP '96, IEEE Computer Society, 1996, pp. 3326–3329.

[26] University of Michigan and University of Colorado, *"sym-panalyzer2.0 reference manual"*, "`http://www.eecs.umich.edu/~panalyzer/pdfs/Sim-Panalyzer2.0_ReferenceManual.pdf`", 2011.

[27] Timothy Osmulski, Jeffrey T. Muehring, Brian Veale, Jack M. West, Hongping Li, Sirirut Vanichayobon, Seok-Hyun Ko, John K. Antonio, and Sudarshan K. Dhall, *A probabilistic power prediction tool for the xilinx 4000-series fpga*, Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing (London, UK), IPDPS '00, Springer-Verlag, 2000, pp. 776–783.

[28] S. Arash Ostadzadeh, Roel Meeuws, Carlo Galuzzi, and Koen Bertels, *Quad - a memory access pattern analyser*, ARC, 2010, pp. 269–281.

[29] Enrique I. Oviedo, *Software engineering metrics i*, McGraw-Hill, Inc., New York, NY, USA, 1993, pp. 52–65.

[30] Sandro Penolazzi, Luca Bolognino, and Ahmed Hemani, *Energy and performance model of a sparc leon3 processor*, Proceedings of the 2009 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools (Washington, DC, USA), DSD '09, IEEE Computer Society, 2009, pp. 651–656.

[31] Kara K. W. Poon, Steven J. E. Wilton, and Andy Yan, *A detailed power model for field-programmable gate arrays*, ACM Trans. Des. Autom. Electron. Syst. **10** (2005), 279–302.

[32] Jan M. Rabaey, *Digital integrated circuits: a design perspective*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[33] Akshaye Sama, J. F. M. Theeuwen, and M. Balakrishnan, *Speeding up power estimation of embedded software*, Proceedings of the 2000 international symposium on Low power electronics and design (New York, NY, USA), ISLPED '00, ACM, 2000, pp. 191–196.

[34] L. Shang and N.K.Jha, *High-level power modeling of cplds and fpgas*, Proceedings of the International Conference on Computer Design: VLSI in Computers & Processors (Washington, DC, USA), IEEE Computer Society, 2001, pp. 46–.

[35] Stefan Steinke, Markus Knauer, Lars Wehmeyer, and Peter Marwedel, *An accurate and fine grain instruction-level energy model supporting software optimizations*, in

Proc. Int. Wkshp Power and Timing Modeling, Optimization and Simulation (PATMOS, 2001.

[36] Richard W. Stevens and Stephen A. Rago, *Advanced programming in the unix(r) environment (2nd edition)*, Addison-Wesley Professional, 2005.

[37] T. K. Tan, A. K. Raghunathan, G. Lakishminarayana, and N. K. Jha, *High-level software energy macro-modeling*, Proceedings of the 38th annual Design Automation Conference (New York, NY, USA), DAC '01, ACM, 2001, pp. 605–610.

[38] Vivek Tiwari, Sharad Malik, and Andrew Wolfe, *Power analysis of embedded software: a first step towards software power minimization*, Proceedings of the 1994 IEEE/ACM international conference on Computer-aided design (Los Alamitos, CA, USA), ICCAD '94, IEEE Computer Society Press, 1994, pp. 384–390.

[39] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K.L.M. Bertels, G.K. Kuzmanov, and E. Moscu Panainte, *The molen polymorphic processor*, IEEE Transactions on Computers (2004), 1363– 1375.

[40] Tajana Šimunić, Luca Benini, and Giovanni De Micheli, *Cycle-accurate simulation of energy consumption in embedded systems*, Proceedings of the 36th annual ACM/IEEE Design Automation Conference (New York, NY, USA), DAC '99, ACM, 1999, pp. 867–872.

[41] Inc Xilinx, *Virtex-5 fpga data sheet: Dc and switching characteristics, ds202 (v5.3)*, `http://www.xilinx.com/support/documentation/data_sheets/ds202.pdf`, May 5, 2010, Website.

[42] Y. D. Yankova, G. Kuzmanov, K.L.M. Bertels, G. N. Gaydadjiev, Y. Lu, and S. Vassiliadis, *Dwarv: Delftworkbench automated reconfigurable vhdl generator*, In Proceedings of the 17th International Conference on Field Programmable Logic and Applications (FPL07), August 2007, pp. 697–701.

[43] P. Zipf, H. Hinkelmann, Lei Deng, M. Glesner, H. Blume, and T.G. Noll, *A power estimation model for an fpga-based softcore processor*, Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on, August 2007, pp. 171 –176.

Appendix 6

6.1 Software Complexity Metrics List

The following table shows the complete list and a description of the SCMs used in this work to characterize power consumption. The SCMs were presented in [24].

Id	Metric	Description
1	AICC	Average Information Content Classification
2	avg.nesting	Average nesting
3	basicblocks	Total number of basic blocks
4	bb.avg.heap.loads	Average heap loads per basic block
5	bb.avg.loads	Average memory loads per basic block
6	bb.avg.operators	Average operators per basic block
7	bb.avg.statements	Average statements per basic block
8	bb.max.heap.loads	Maximum heap loads per basic block
9	bb.max.loads	Maximum memory loads per basic block
10	bb.max.operators	Maximum operators per basic block
11	bb.max.statements	Maximum statements per basic block
12	bitextracts	Total number of bit extract operations
13	bitinserts	Total number of bit inserts operations
14	bits.bitwise	Cumulative number of bits used in bitwise operations
15	bits.constants	Cumulative number of bits of contants in the code
16	bits.fp.alus	Cumulative number of bits used in floating point ALU operations
17	bits.fp.divs	Cumulative number of bits used used in floating point divisions
18	bits.fp.mods	Cumulative number of bits used in floating point modules
19	bits.fp.mults	Cumulative number of bits used in floating points operations
20	bits.heap.loads	Cumulative number of bits used in loads to heap
21	bits.heap.stores	Cumulative number of bits used in stores to heap
22	bits.int.alus	Cumulative number of bits used in integer ALU operations
23	bits.int.comps	Cumulative number of bits used in integer comparisons
24	bits.int.constdivs	Cumulative number of bits used in integer divisions of constants
25	bits.int.constmods	Cumulative number of bits used in integer module of constants
26	bits.int.constmults	Cumulative number of bits used in integer multiplication of constants
27	bits.int.divs	Cumulative number of bitsused in integer divisions
28	bits.int.mods	Cumulative number of bits used in integer modules

continued on next page ...

Id	Metric	Description
29	bits.int.mults	Cumulative number of bits used in integer multiplications
30	bits.int.tests	Cumulative number of bits used in integer tests
31	bits.loads	Cumulative number of bits used in memory loads
32	bits.locals	Cumulative number of bits used in local variables
33	bits.negate	Cumulative number of bits used in negations operation
34	bits.parameters	Cumulative number of bits used in the arguments of the function
35	bits.stores	Cumulative number of bits used in memory stores
36	bits.type.conversions	Cumulative number of bits used in type conversions
37	bits.used.globals	Cumulative number of bits used in operations with globals variables
38	bitwise	Total number of bitwise operations
39	constants	Total number of constants
40	cum.nesting	Cumulative number of nesting deepness
41	cyclomatic	Cyclomatic metric [21]
42	elshof.data.flow	Data flow complexity metric
43	executed.operators	Total number of executed operators
44	executed.statements	Total number of statements executed
45	fp.alus	Total number of floating point ALU ops
46	fp.divs	Total number of floating point divisions
47	fp.mods	Total number of floating point modules
48	fp.mults	Total number of floating point multiplications
49	function.calls	Total number of function calls
50	heap.loads	Total number of loads to heap
51	heap.stores	Total number of stores to heap
52	int.alus	Total number of integer ALU ops
53	int.barrelshifts	Total number of integer barrel shifts
54	int.comps	Total number of integer comparisons
55	int.constantshifts	Total number of integer constants shifting
56	int.constdivs	Total number of integer division of constants
57	int.constmods	Total number of integer modules of constants
58	int.constmults	Total number of integer multiplication of constants
59	int.divs	Total number of integer divisions
60	int.mods	Total number of integer modules
61	int.mults	Total number of integer multiplications
62	int.tests	Total number of integer tests
63	loads	Total number memory loads
71	oviedo	Oviedo metric [29]
72	parameters	Number of parameters (arguments) of the function
73	pointers	Number of pointers

continued on next page ...

Id	Metric	Description
74	returns.value	Binary metric that shows if the function returns a value
75	scope.number	Number of scopes
76	scope.ratio	Ratio in the scopes
77	statements	Number of statements
78	stores	Number of memory stores
79	type.conversions	Number of type conversions
80	uOperands	Number of micro-operands
81	uOperators	Number of micro-operators
82	used.globals	Number of global variables used

Table 6.1: Quipu Software Complexity Metrics

6.2 Configuration parameters of Sim-Panalyzer

In the Listing 6.1, we present the default configuration parameters of a StrongARM processor. This configuration data is given in the source code of sim-panalyzer 2.0.3, available in [26].

Listing 6.1: Default configuration parameters of a StrongARM processor provided by sim-panalyzer

```
1  #
2  # SA-1 core sim-outorder configuration
3  #
4
5  -seed                              1
6  -fetch:ifqsize                     8
7  -fetch:mplat                       9999
8  -fetch:speed                       1
9  -bpred                          nottaken
10 -decode:width                      2
11 -issue:width                       2
12 -commit:width                      2
13 -issue:inorder                   true
14 -issue:wrongpath                 true
15 -ruu:size                          4
16 -lsq:size                          4
17 -lsq:perfect                     false
18 -cache:dl1              dl1:16:32:32:f
19 -cache:dl1lat                      1
20 -cache:dl2                       none
21 -cache:il1              il1:16:32:32:f
22 -cache:il1lat                      1
23 -cache:il2                       none
24 -cache:flush                     false
25 -cache:icompress                 false
26 -mem:lat                         64 1
27 -mem:width                         4
28 -mem:pipelined                   false
```

```
-tlb:itlb             itlb:32:4096:32:f
-tlb:dtlb             dtlb:32:4096:32:f
-tlb:lat                          30
-res:ialu                          2
-res:imult                         1
-res:memport                       1
-res:fpalu                         1
-res:fpmult                        1
-bugcompat                    false
-panalyzer:aio  aio:a:200:o:3.3:5:10:1
-panalyzer:btb          btb:a:200:1:1:1
-panalyzer:clock   clock:a:200:n:250:3
-panalyzer:dio  dio:a:200:b:3.3:5:10:1
-panalyzer:dl1          dl1:a:200:1:1:1
-panalyzer:dtlb       dtlb:a:200:1:1:1
-panalyzer:fprf           fprf:a:200:1
-panalyzer:il1          il1:a:200:1:1:1
-panalyzer:irf              irf:a:200:1
-panalyzer:itlb       itlb:a:200:1:1:1
-panalyzer:dl2          dl2:a:200:1:1:1
-panalyzer:ras              ras:a:200:1
-panalyzer:logic logic:a:200:1.8:Static:30000:4:1:4
```

6.3 Energy and Power relationship

In the book Digital Integrated Circuits a Design Perspective [32], Rabaey defines the concepts of power and energy consumption for CMOS devices. These concepts are closely related and can be related mathematically using the theory of the dynamic behavior in the CMOS inverter.

The dynamic energy dissipation of a CMOS inverter is physically related to the charge and discharge of the voltage in the load capacitor. This charge and discharge process occurs twice during a switching cycle, consisting of Low-to-High and High-to-Low transitions. The energy consumed during this process can be calculated using the Equation 6.1, where energy(E) is equal to the load capacitance (C_L) times the supply voltage to the square (V_{DD}^2).

$$E = C_L \times V_{DD}^2 \tag{6.1}$$

The energy consumed by a CMOS inverter is associated with a chargedischarge cycle only, and this cycle is not necessarily equal to the frequency of the clock.

Now that we define the energy, we can relate energy with power consumption using the Equation 6.2 that describes the dynamic power consumption of the CMOS inverter.

$$P_{dyn} = E \times switching\_activity = C_L \times V_{DD}^2 \times switching\_activity \tag{6.2}$$

The term *switching activity*, presented in the previous equation, refers to the transitions from Low-to-High and High-to-Low that occurs in one second in the CMOS inverter. For complex systems, such as a GPP or an FPGA, the *switching activity* is difficult to calculate because depends on the implementation of the system, the inputs provided to

the system, and in general, the dynamic behavior of the system. However, it is possible to relate the *switching activity* with the clock frequency of a system. It is important to notice that the clock frequency is not necessarily equal to the *switching activity*, because it is possible that during a clock cycle of a design some CMOS inverters do switch and others not. Therefore, we can use a transition probability of the signals, in a design, to express how probable is that a signal switches from High-to-Low and Low-to-High during a clock cycle of the system. Using the transition probability P_{0-1} we can express the dynamic power using the Equation 6.3

$$P_{dyn} = C_L \times V_{DD}^2 \times P_{0-1} \times f \tag{6.3}$$

The variable f represents the clock frequency. Based on this formula, we can define the concept of *effective capacitance*, that is calculated multiplying the load capacitance times the transition probability (P_{0-1}). The *effective capacitance* is defined as the average capacitance switched every clock cycle, and can be expressed with Equation 6.4.

$$C_{eff} = C_L \times P_{0-1} \tag{6.4}$$

Using the definition of *effective capacitance*, we can define the dynamic power as presented in Equation 6.5.

$$P_{dyn} = C_{eff} \times V_{DD}^2 \times f \tag{6.5}$$

Finally, we can use Equation 6.5 to relate energy and dynamic power consumption, as presented in Equation 6.6.

$$E = C_{eff} \times V_{DD}^2 = \frac{P_{dyn}}{f} = P_{dyn} \times T \tag{6.6}$$

Where T, is the clock period.

6.4 The kernel library

The following table, presents a summary of the applications and kernels contained in the *kernel library*.

Kernel Name	Kernel description	Application description	Application domain
g721_body	CCITT G.721 32Kbps ADPCM coder	ANSI-C language reference implementations of the CCITT by Sun Microsystems	Multimedia
mdct_butterfly_8	Normalized modified discrete cosine transform - 8 point butterfly	Vorbis a general purpose audio and music encoding format contemporary to MPEG-4's AAC and TwinV. libvorbis-1.2.0	Multimedia
mdct_butterfly_16	Normalized modified discrete cosine transform - 16 point butterfly		Multimedia
CrossProduct3	Cross product implementation of an integer vector of 3 elements	CrossProduct library with cross product implementations using floating point	Mathematics

continued on next page ...

Kernel Name	Kernel description	Application description	Application domain
CrossProduct7	Cross product implementation of a an integer vector of 7 elements	and integer numbers	Mathematics
bytesum	Addition of high byte and low byte of a long number	Functions library with implementations of different mathematical operations. V.1.	Mathematics
polynomialbin	Subtitution of independant variable in a polynomial equation		Mathematics
intersect_triangle	Intersect triangle algorithm	Testbench for intersect triangle algorithm	Mathematics
intmatmult3x3	Matrix multiplication of 3x3 matrix	Testbench application for matrix operations	Mathematics
intmatmult4x4	Matrix multiplication of 4x4 matrix		Mathematics
mseq	m-sequence sequences generator (small set and large set)	Spreading sequences generators for CDMA mobile communications	Error Correction Code (ECC)
apply_butterflies	Stream decoder for rate 1/3 K=7	KA9Q Viterbi decoder V3.0.1	ECC
Hamming1_unrolled	Unrolled function to get the hamming distance of two unsigned short numbers. Version 1	Testbench of the the Hamming distance, a metric determined by the sum of the absolute bitwise difference of two operands.	ECC
Hamming2_unrolled	Unrolled function to get the hamming distance of two unsigned short numbers. Version 2	Testbench of the the Hamming distance, a metric determined by the sum of the absolute bitwise difference of two operands.	ECC
gost_encrypt	The GOST 28147-89 cipher - encoder	Libmcrypt is a thread-safe library providing a uniform interface to access several block and stream encryption algorithms. (libmcrypt-2.5.7)	Cryptography
gost_decrypt	The GOST 28147-89 cipher - decoder		
twofish_encrypt	Implementation of Twofish algorithm by Bruce Schneier, et.al. - encoder Written by Dr B R Gladman		
twofish_decrypt	Implementation of Twofish algorithm by Bruce Schneier, et.al.- decoder Written by Dr B R Gladman		
serpent_encrypt	Implementation of the Serpent algorithm by R.Anderson, E.Biham and L.Knudsen- encoder Written by Dr B R Gladman		
serpent_decrypt	Implementation of the Serpent algorithm by R.Anderson, E.Biham and L.Knudsen- decoder Written by Dr B R Gladman		
cast256_encrypt	Implementation of the CAST-256 algorithm by Carlisle Adams - encoder written by Dr B R Gladman		
cast256_decrypt	Implementation of the CAST-256		

continued on next page ...

Kernel Name	Kernel description	Application description	Application domain
	algorithm by Carlisle Adams decoder written by Dr B R Gladman		
loki97_encrypt	Implementation of the LOKI97 algorithm by Brown and Pieprzyk encoder written by Dr B R Gladman		
loki97_decrypt	Implementation of the LOKI97 algorithm by Brown and Pieprzyk decoder written by Dr B R Gladman		
cast128_encrypt	Implementation of the CAST-128 algorithm by Niels Muller - encoder Written by Steve Reid		
cast128_decrypt	Implementation of the CAST-128 algorithm by Niels Muller - decoder Written by Steve Reid		
des_f	Implementation of the Feistel function of the Data Encryption Standard (DES)		
MD5Transform	This function alters an existing MD5 hash to reflect the addition of 16 longwords of new data.	The mhash library provides an easy way to access strong hashes such as MD5, SHA1 and other algorithms. (mhash-0.9.6)	Cryptography
sha_transform	Perform the SHA transformation.		
MD4Transform	The core of the MD4 algorithm.		
hw_ripemd128_transform	RIPEMD-128 is a plug-in substitute for RIPEMD with a 128-bit result		
hw_ripemd160_transform	RIPEMD-160 is a 160-bit cryptographic hash function designed by H.Dobbertin, A.Bosselaers, and B.Preneel.		
hw_ripemd256_transform	RIPEMD-256 is an extension of RIPEMD-128		
hw_ripemd320_transform	RIPEMD-320 is an extension of RIPEMD-160		
havalTransform3	HAVAL - the one-way hashing algorithm with 3 passes		
bitreversal2_unrolled	Code to perform in-situ index bit-reversalfor danlan (module ifft.c) - unrolled version	Testbench of bitreversal function based on Sculptor-3.3 library	DSP
complex_div	Division of complex numbers	Functions library with	Mathematics

continued on next page ...

Kernel Name	Kernel description	Application description	Application domain
complex_mult	Multiplication of complex numbers	implementationsof different mathematical operations. V.2.	Mathematics
max	Returns the maximum of 4 integer numbers	NW Implements the Needleman-Wunsch algorithm for global nucleotide sequence alignment, written by R.Muertter 9/5/2006	Bio- informatics

Table 6.2: Summary of kernel library

6.5 Experiment StrongARM.SimPanalyzer.GPP.B.1 - energy relationship

The Figures 6.1, 6.2, 6.3, 6.4, 6.5, and 6.6 show the relation ship between the energy and the parameters used for modelling.

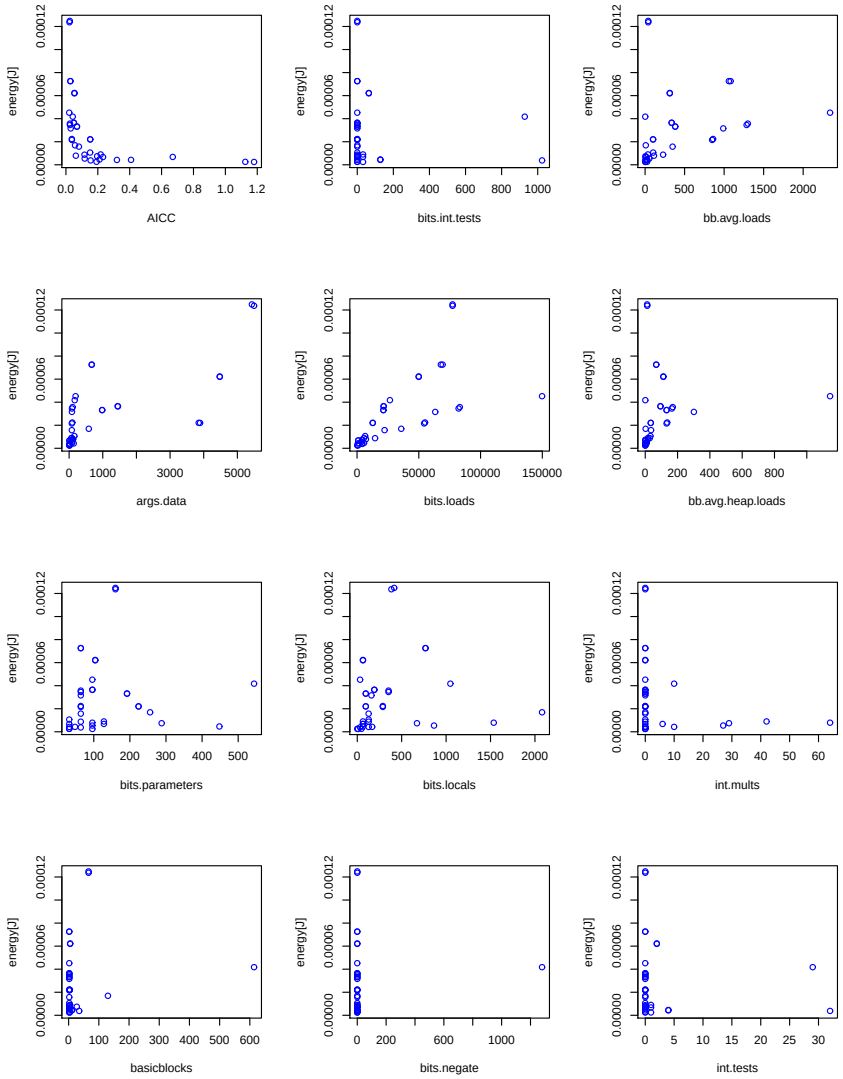


Figure 6.1: Relation between energy and modelling parameters, graph1

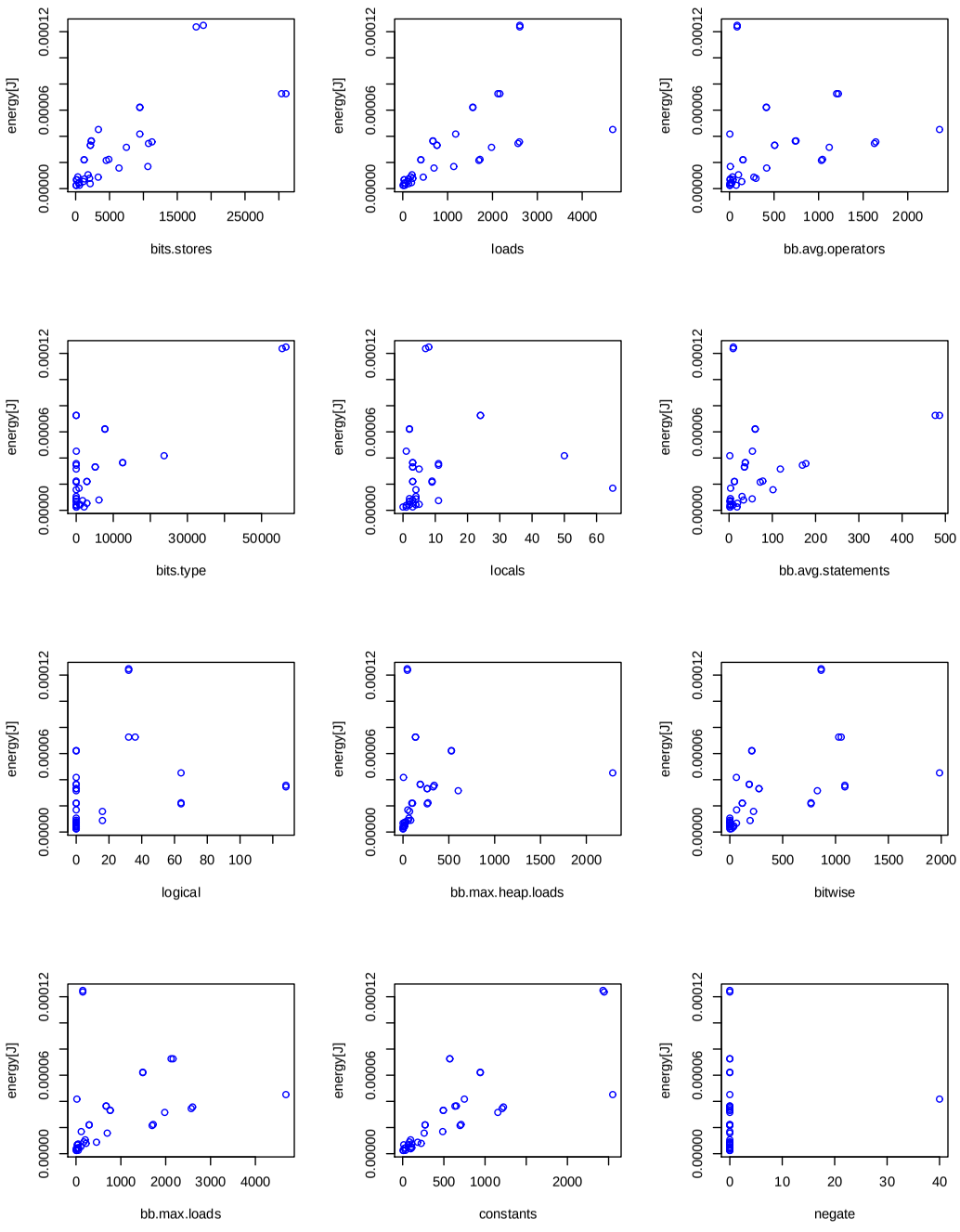


Figure 6.2: Relation between energy and modelling parameters, graph2

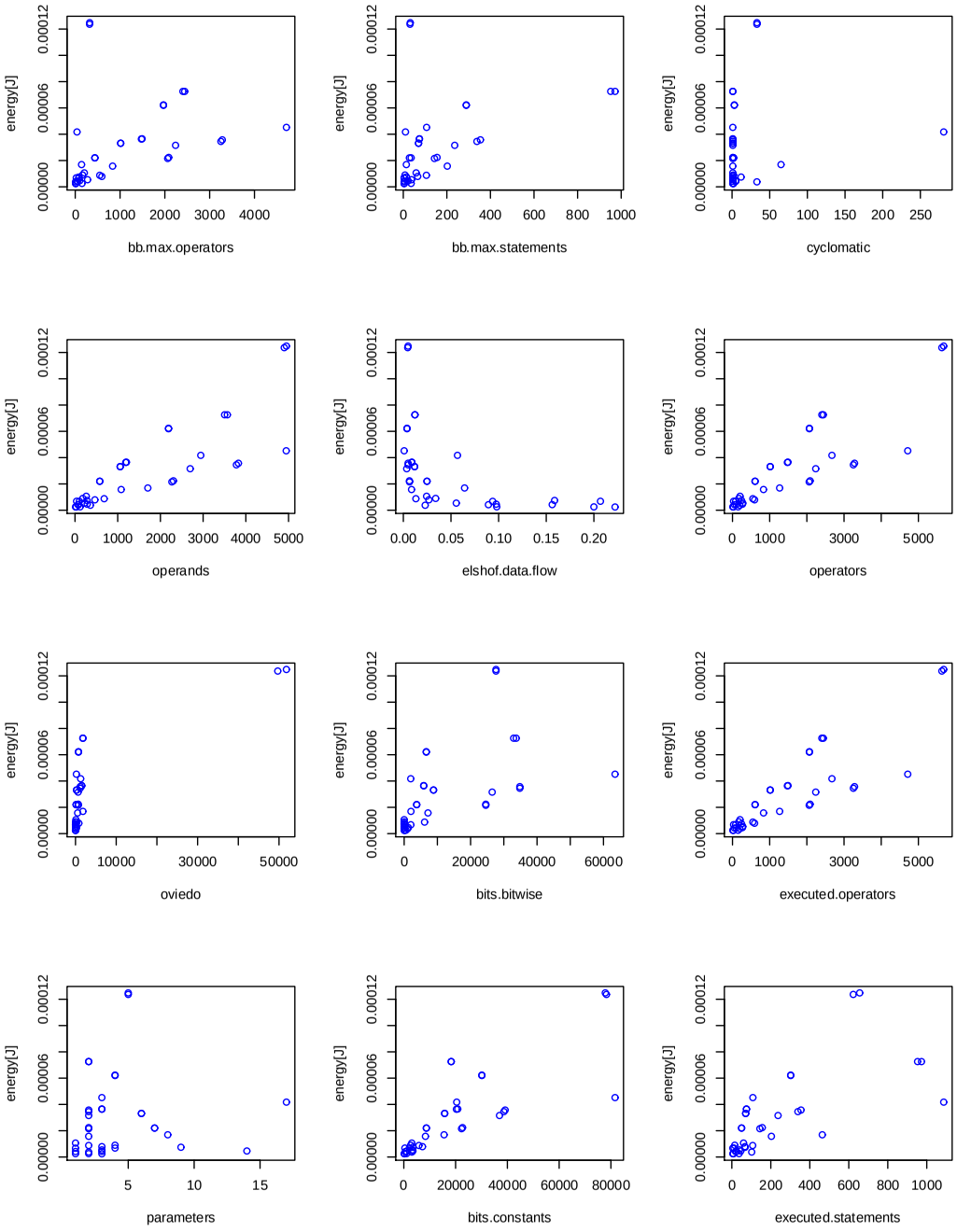


Figure 6.3: Relation between energy and modelling parameters, graph3

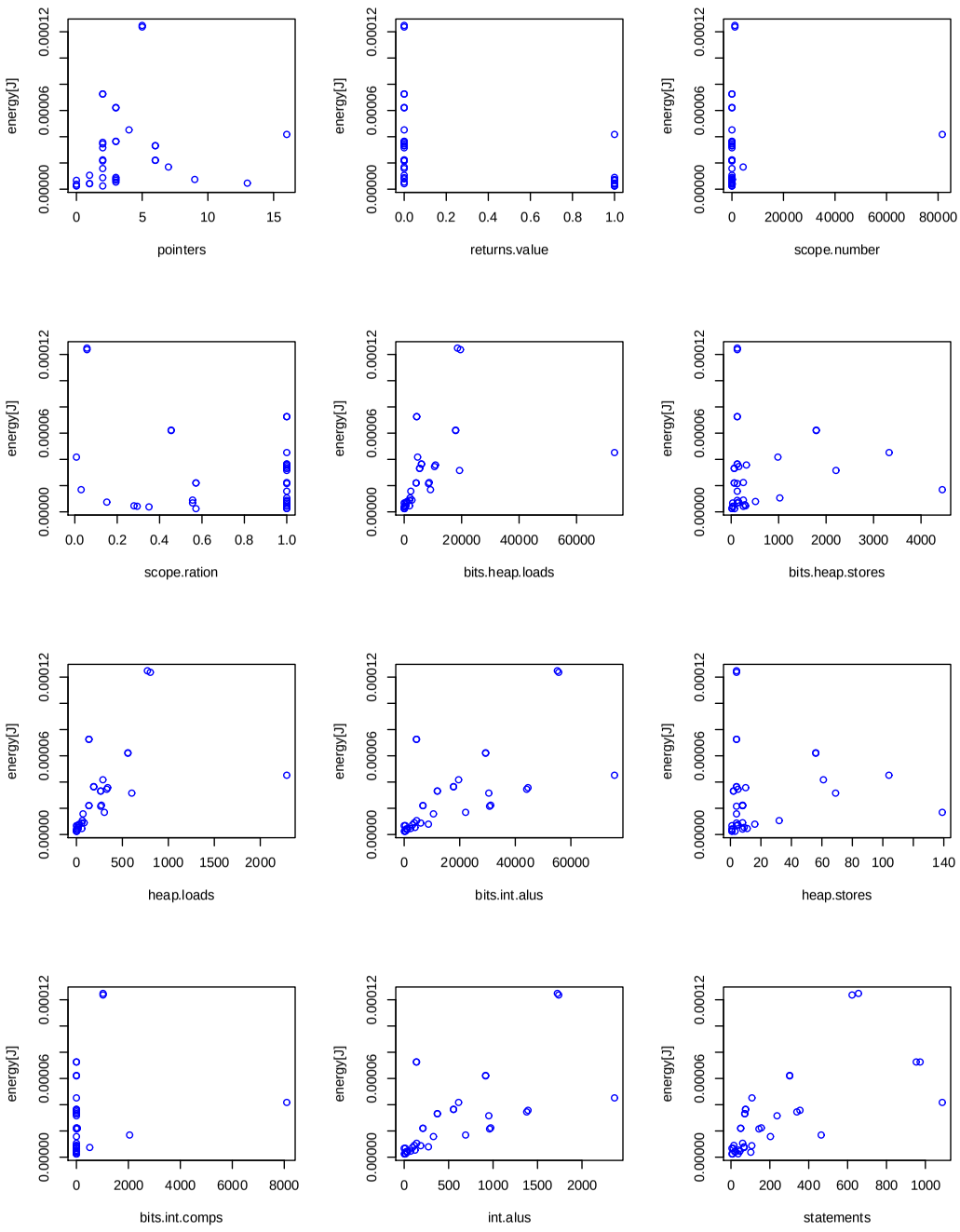


Figure 6.4: Relation between energy and modelling parameters, graph4

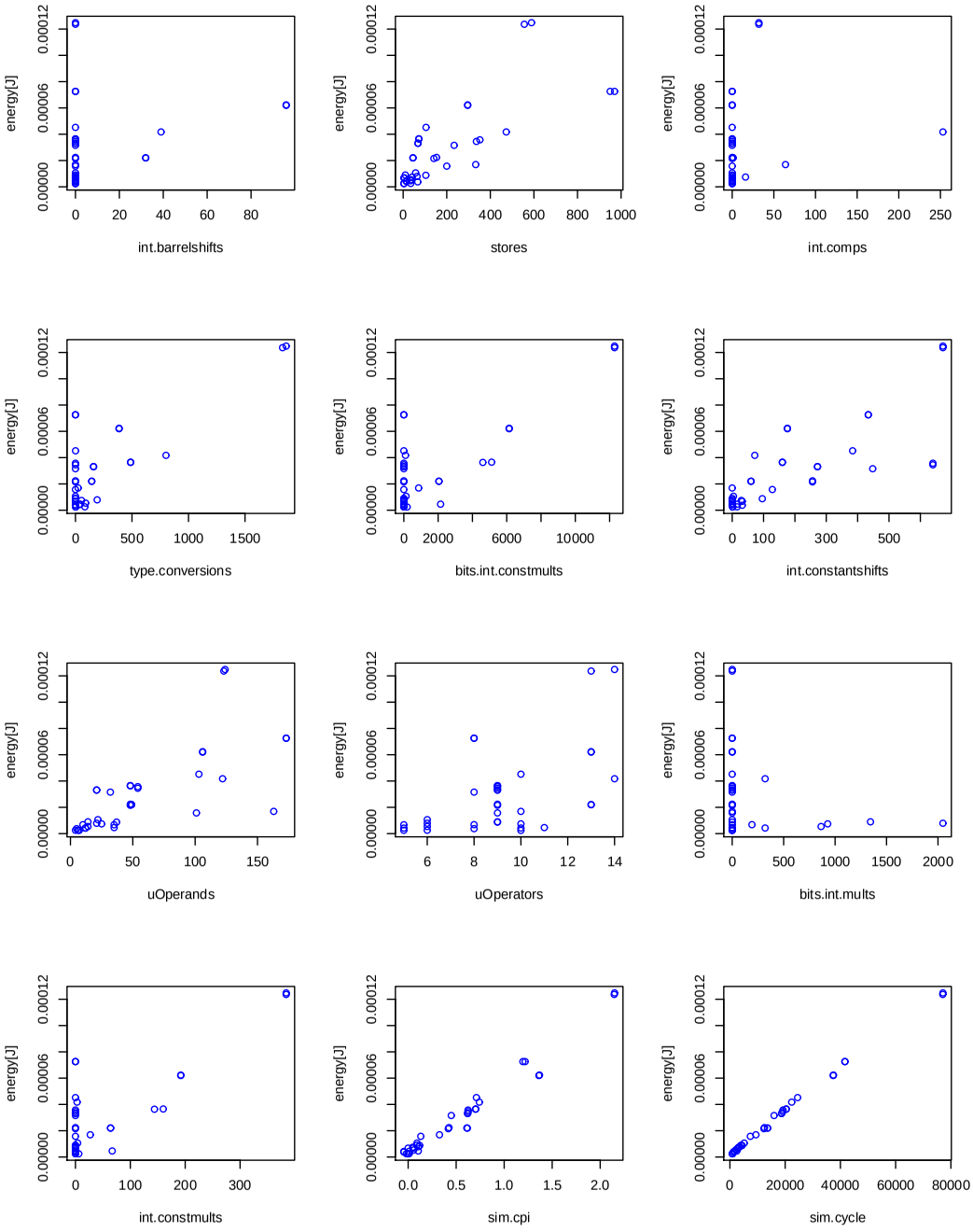


Figure 6.5: Relation between energy and modelling parameters, graph5

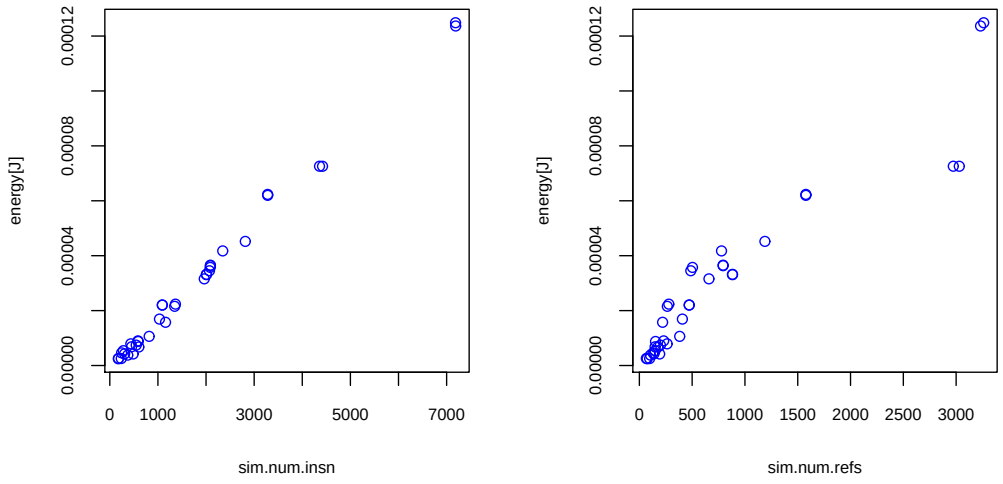


Figure 6.6: Relation between energy and modelling parameters, graph6

6.6 Experiment StrongARM.SimPanalyzer.GPP.B.1 - Power relationship

The Figures 6.7, 6.8, 6.9, 6.10, 6.11, and 6.12 show the relation ship between the power consumption in a kernel, and the predictors used in the modelling process.

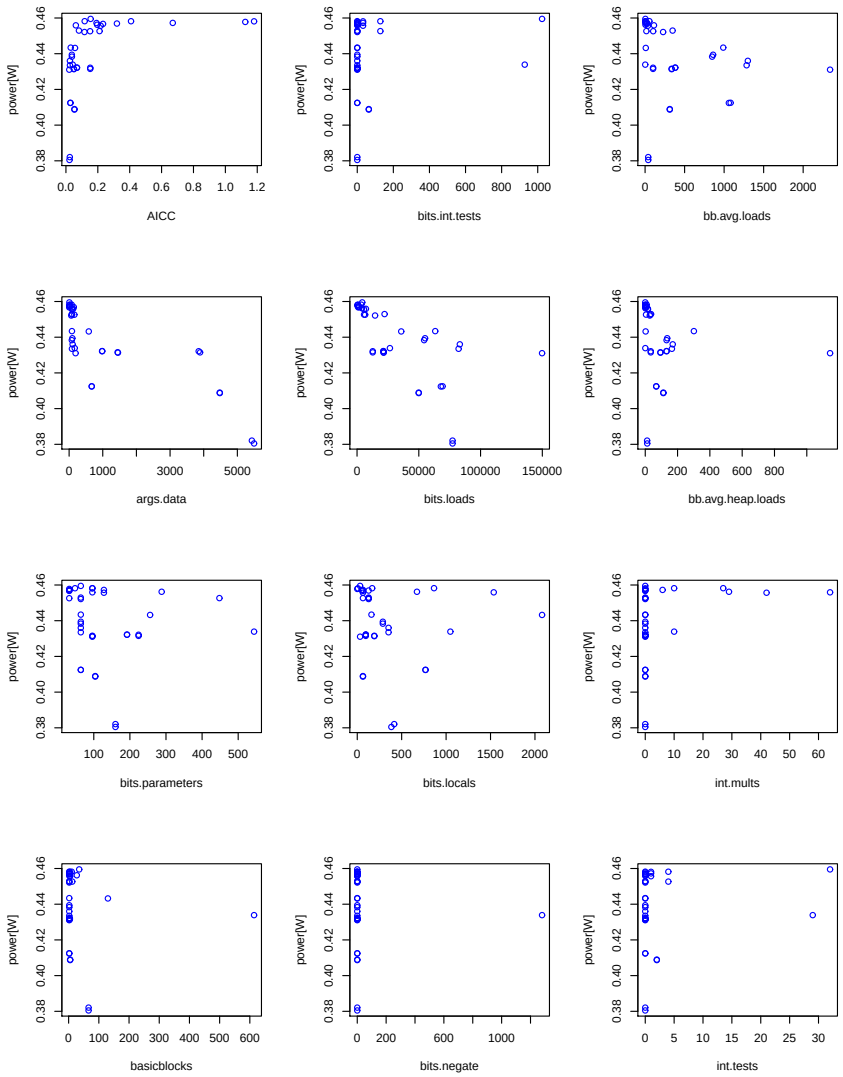


Figure 6.7: Relation between power and the modelling predictors, graph1

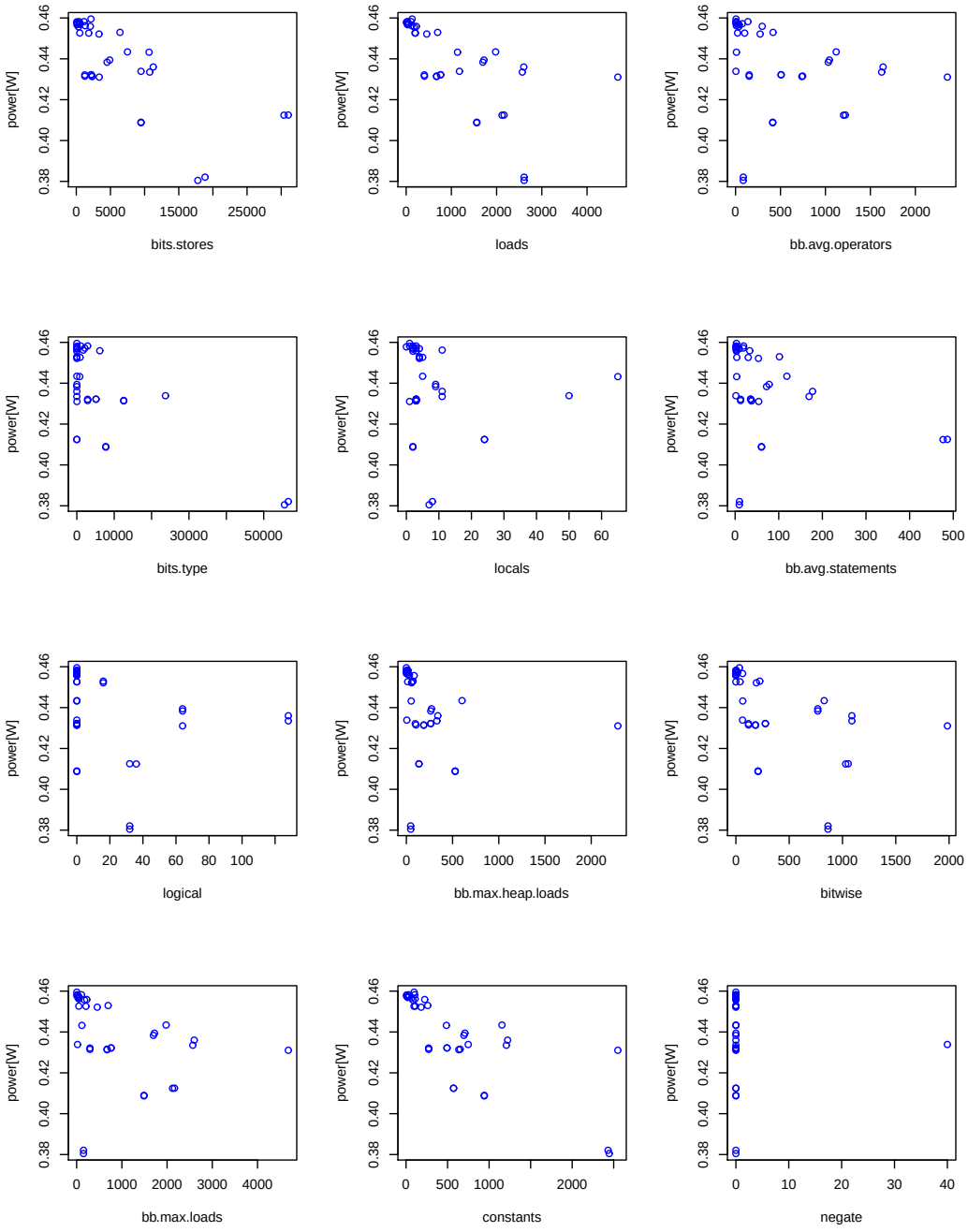


Figure 6.8: Relation between the power and the modelling predictors, graph2

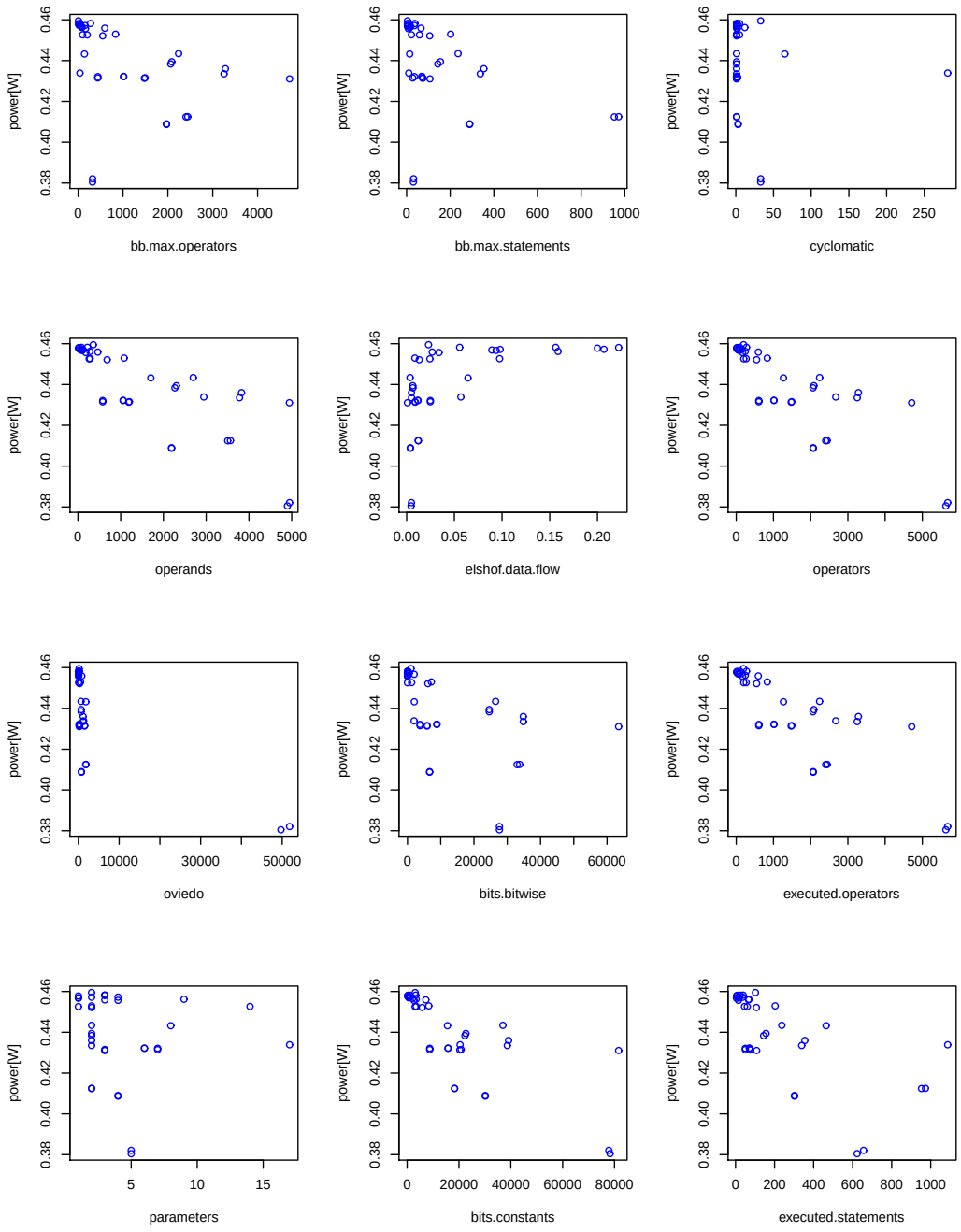


Figure 6.9: Relation between power and the modelling predictors, graph3

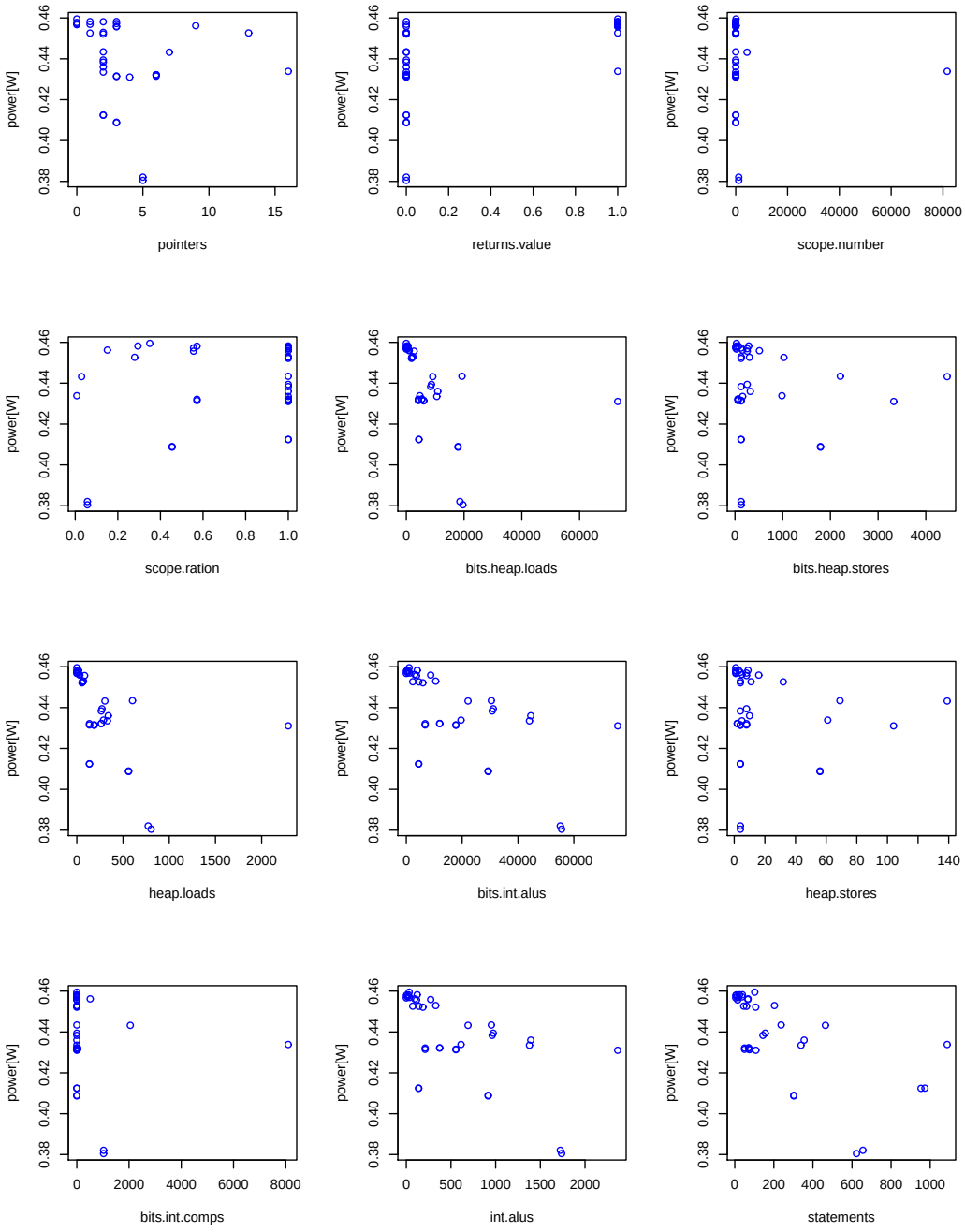


Figure 6.10: Relation between power and the modelling predictors, graph4

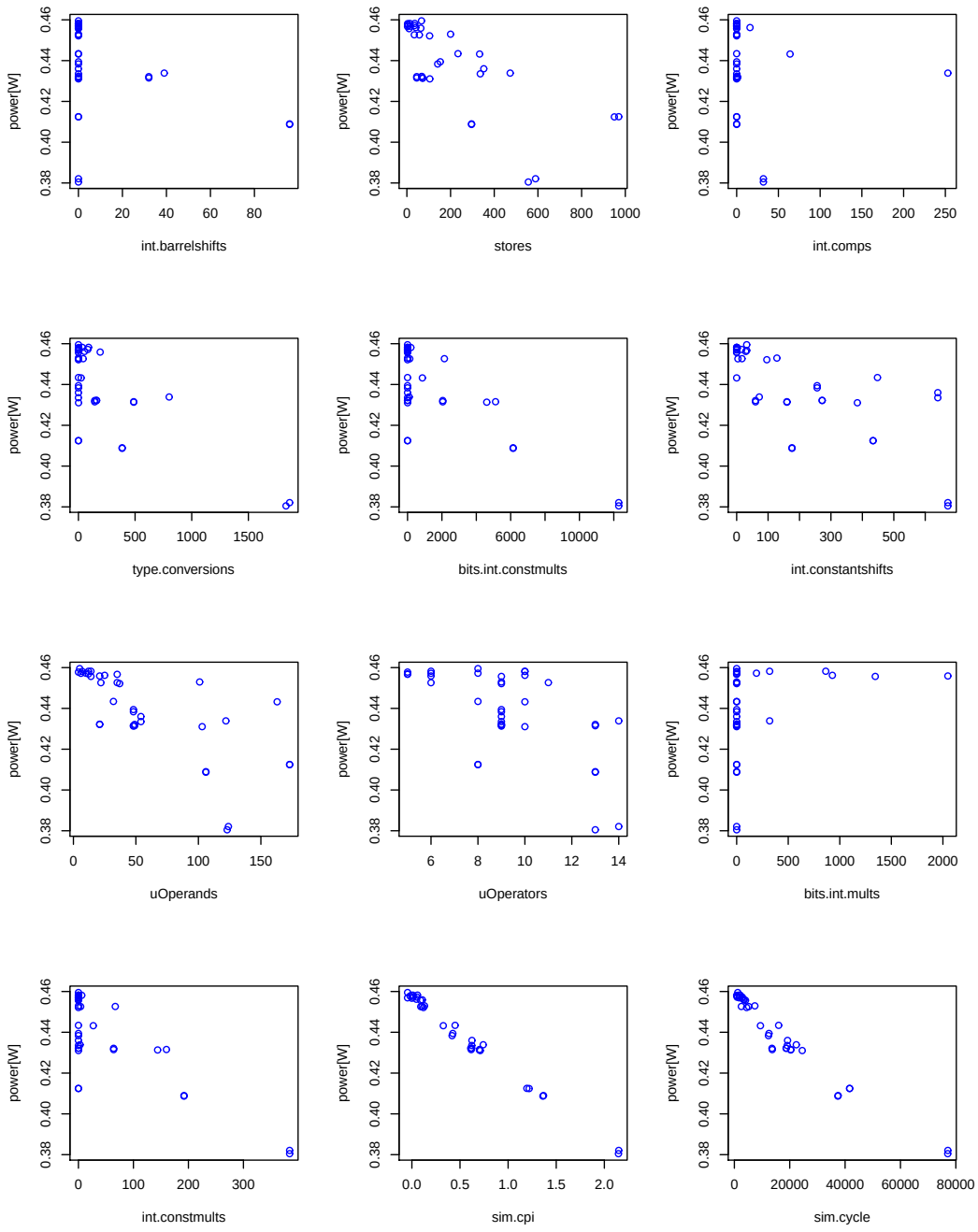


Figure 6.11: Relation between power and the modelling predictors, graph5

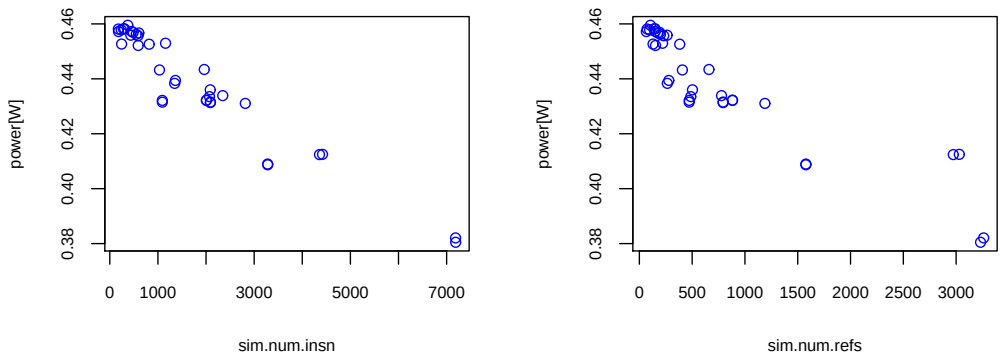


Figure 6.12: Relation between power and the modelling predictors, graph6

6.7 Models generated in the experiment VIRTEX5.Xpwr.FPGA.B.2

In Figures 6.13, 6.14, 6.15, 6.16, 6.17, 6.18, 6.20, and 6.19, we present the models created in the experiment VIRTEX5.Xpwr.FPGA.B.2.

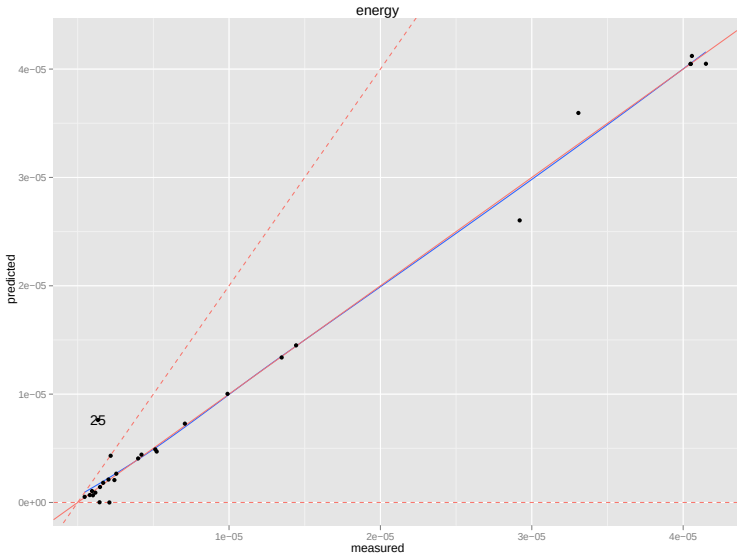


Figure 6.13: Energy model for the Virtex 5 FPGA, with static predictors

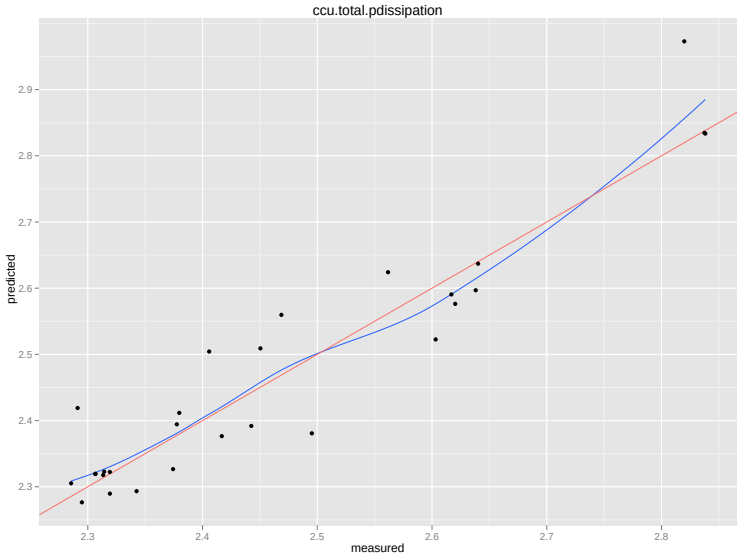


Figure 6.14: Total power consumption model for the Virtex 5 FPGA, with static predictors

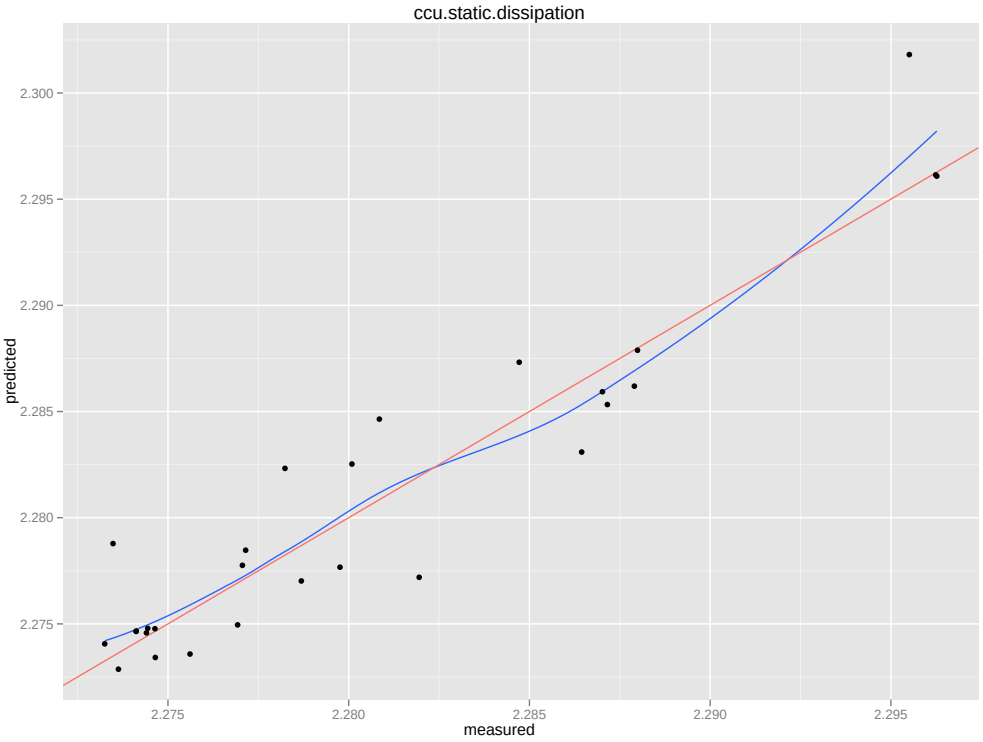


Figure 6.15: Static power consumption model for the Virtex 5 FPGA, with static predictors

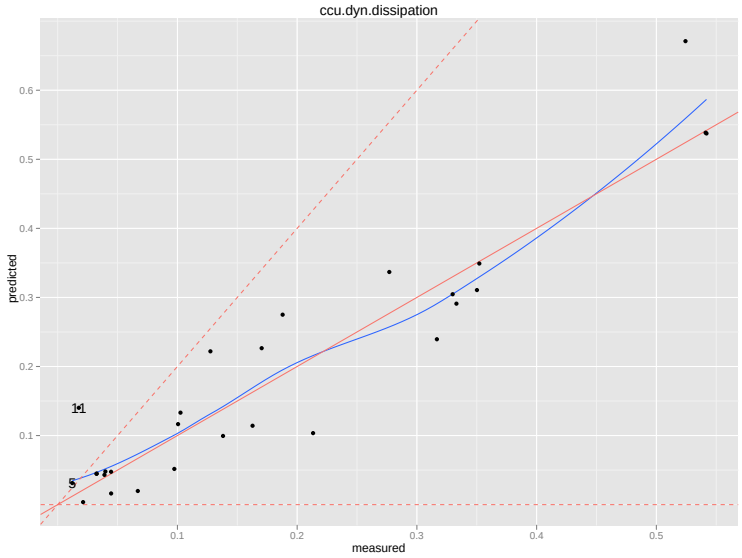


Figure 6.16: Dynamic power consumption model for the Virtex 5 FPGA, with static predictors

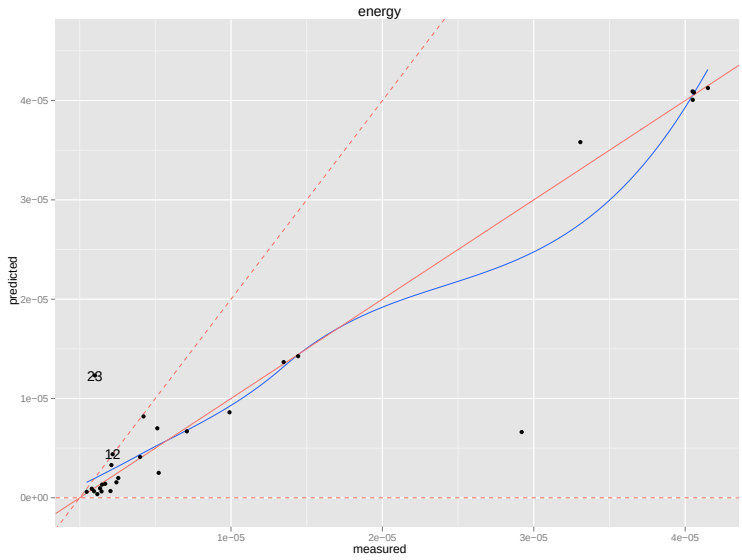


Figure 6.17: Energy model for the Virtex 5 FPGA, with static & dynamic predictors

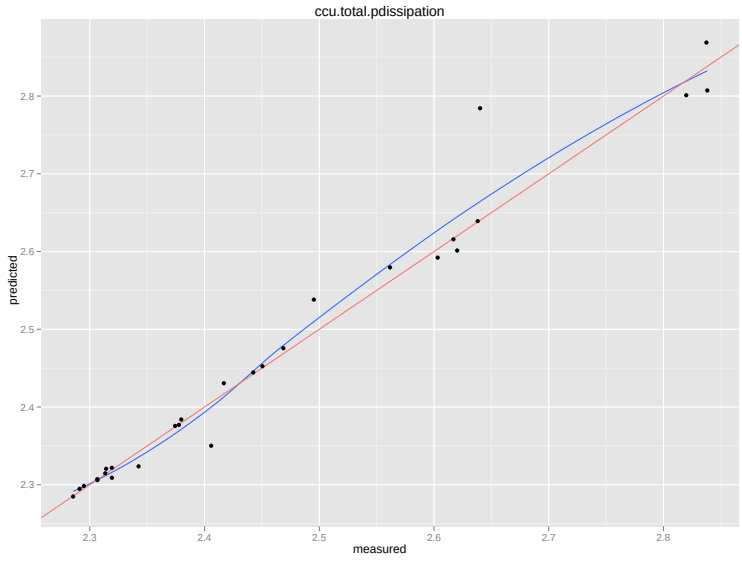


Figure 6.18: Total power consumption model for the Virtex 5 FPGA, with static & dynamic predictors

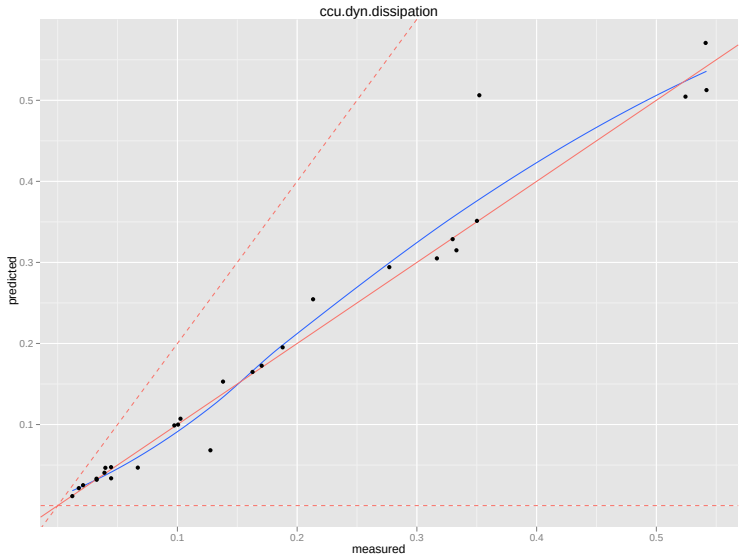


Figure 6.19: Dynamic power consumption model for the Virtex 5 FPGA, with static & dynamic predictors

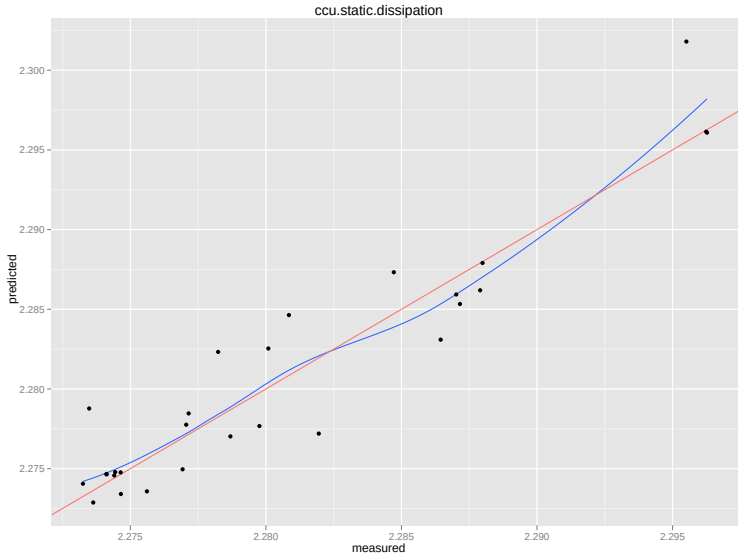


Figure 6.20: Static power consumption model for the Virtex 5 FPGA, with static & dynamic predictors