Synthesis and characterisation of MOVE configurations

A new processor generator and a modeller proposal

Marnix Arnold

Laboratory of Computer Architecture
and Digital Techniques (CARDIT)
Faculty of Electrical Engineering
Delft University of Technology
P.O. Box 5031, NL-2600 GA Delft,
The Netherlands

June 21, 1995

Delft University of Technology
Faculty of Electrical Engineering

Type	:	Master's Thesis
Number of pages	:	145
Date	:	June 21, 1995
Lab./Dept.	:	Laboratory of Computer Architecture and Digital Techniques (CARDIT)
Codenumber	:	1-68340-28(1995)14
Author	:	Marnix Arnold
Title	:	**Synthesis and characterisation of MOVE configurations** A new processor generator and a modeller proposal
Supervisor	:	Prof. Dr. Ir. A. J. van de Goor
Mentor	:	Drs. H.Corporaal

Abstract

Due to the flexibility and scalability of the MOVE processor architecture, there is an enormous number of possible processor configurations. When developing a processor, all these configurations must be evaluated (characterised) in order to be able to make a well-considered choice. Given the current MOVE development framework, this takes an unacceptable amount of time.

To shorten the design time for MOVE processors, a MOVE Processor Generator was developed, based on an earlier model (the MOVEASAINT) that uses the ASA design tool. The new processor generator uses languages and tools that are currently considered to be the industry standard (VHDL, Synopsys, Cadence).

In order to speed up the characterisation of a series of MOVE configurations, a modeler structure has been conceived. It is based on the fact that for designs resulting from different configurations, parts may actually be the same, allowing re-use of previously obtained, partial characterisation data. This means that only previously unknown data must be computed with the processor generator instead of the entire design each time. Previous research (the MOVEmate project) suggests that a considerable evaluation time reduction is possible for large sets of configurations.

Keywords

Transport Triggered Architectures, MOVE, synthesis, characterisation, modelling

Preface

It goes without saying that it is impossible to complete a Master's research project all by yourself. I'd like to take this opportunity to thank all the people who have helped me in one way or another during the past year:

Henk Corporaal for providing guidance on the long road to completion of this project.
Reinoud Lamberts for many an interesting discussion about EDA tools, industrial bureaucracy and Bill Gates, and of course for helping me gain insight into the complex matter of MOVE hardware synthesis. His feedback has been invaluable. Marcel van der Laan for shedding light on those tricky modelling issues.
Antoon Frehe and Tobias Nijweide for supplying UNIX support and lots of brute force computing power. Their systems' loads will be considerably lighter the next months, not having to bear my colossal synthesis jobs. Antoon deserves an honourable mention for standing by me in my battles with software bugs, licence trouble and many other problems I don't want to think about anymore.
Dr. Nouta for doing his best to provide me with all the software tools I needed, and of course for helping me with my internship arrangements.
Wiebe Cnossen for building the Latex styles with which this report was formatted in no time at all.
My fellow students who must have wondered what was the matter with me, not joining them in the student computer room but keeping to the 14th floor instead. You wouldn't have liked me to hog all the processor time downstairs, would you?
My friends, some of who had to go to Brussels and Toulouse without me because I was too busy slavering over my thesis.
And, last but certainly not least, my parents, whose support I've always been able to count on. This despite the fact that they often might not have had a clue what I was talking about whenever I tried to explain what it was that I was doing.

Delft, June 1995.

Synthesis and characterisation of MOVE configurations

A new processor generator and a modeller proposal

Abstract

Due to the flexibility and scalability of the MOVE processor architecture, there is an enormous number of possible processor configurations. When developing a processor, all these configurations must be evaluated (characterised) in order to be able to make a well-considered choice. Given the current MOVE development framework, this takes an unacceptable amount of time.

To shorten the design time for MOVE processors, a MOVE Processor Generator was developed, based on an earlier model (the MOVEASAINT) that uses the ASA design tool. The new processor generator uses languages and tools that are currently considered to be the industry standard (VHDL, Synopsys, Cadence).

In order to speed up the characterisation of a series of MOVE configurations, a modeler structure has been conceived. It is based on the fact that for designs resulting from different configurations, parts may actually be the same, allowing re-use of previously obtained, partial characterisation data. This means that only previously unknown data must be computed with the processor generator instead of the entire design each time. Previous research (the MOVEmate project) suggests that a considerable evaluation time reduction is possible for large sets of configurations.

Contents

List of Figures

List of Tables

1 Introduction

The first two sections in this chapter are meant to give some background information about the MOVE processor architecture. They have been kept brief intentionally, as plenty of other material has already been written on the subject ([CM91a], [CM92], [Cor94a], [Cor94b], [HC94], [Sch91]). The third and fourth sections address the problems around which this report is centered. In the final section, an overview of the rest of the report is given.

1.1 The MOVE microprocessor architecture

The MOVE architecture [CM92] was developed at the Computer Architecture Laboratory. It derives its name from the fact that the one instruction it can carry out is a move. Being a so-called Transport Triggered Architecture or TTA [Cor94a], [Cor94b], it is composed of a transport network which connects several functional units with each other. Operations are carried out by moving data between functional units over the MOVE buses, in parallel (fig. 1.1). Data arriving at the inputs of a functional unit is automatically processed and the result is returned to the transport network.
The reason for this approach is the fact that in conventional processor architectures like CISC, RISC and super-scalar [Goo89], [HP90], data transport often turns out to be the performance bottleneck. In the MOVE architecture, the possibility (or indeed, necessity) to specify data transports explicitly makes it easier to exploit the data transport capacity to the fullest.

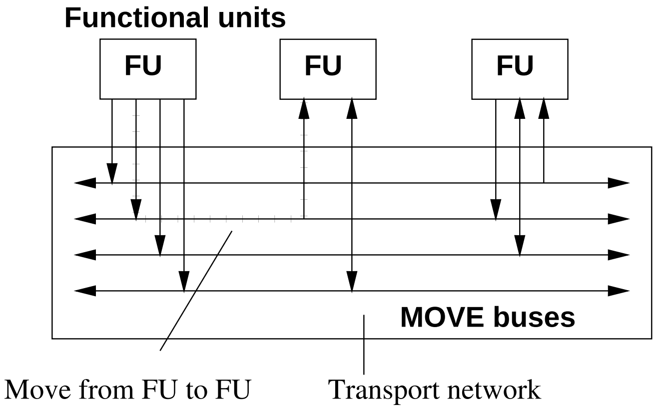


Figure 1.1: A move instruction in a TTA.

An interesting feature of the MOVE architecture is its scalability [CM91a]. Functional units are connected to the transport network in a uniform way, which makes it possible to extend the processor's functionality by simply connecting more functional units. The number of transport buses can also easily be varied, according to the demands on transportation capacity.

1.2 The MOVE development framework

To be able to exploit the MOVE architecture's scalability to the fullest, the MOVE development framework [CM91b], [M^+89] has been initiated. It comprises a suite of hardware and software development tools, aimed at rapid, concurrent compiler and processor design (figure 1.2).
The MOVE development framework attempts to find the MOVE processor configuration which is best suited to the user's requirements in terms of application performance on one hand, and cost requirements on the other. This is to be done automatically, without the user having to intervene.

1.3 Problem definition

Finding the most suitable architecture configuration for a given application is not trivial. In order to be able to make a well-considered choice from all possible configurations, information is needed about their area cost, timing, and power consumption characteristics.
This means implementing and analysing a great number of processor designs, which at present would take an unacceptable amount of time. It's therefore necessary to speed up this process as much as possible.

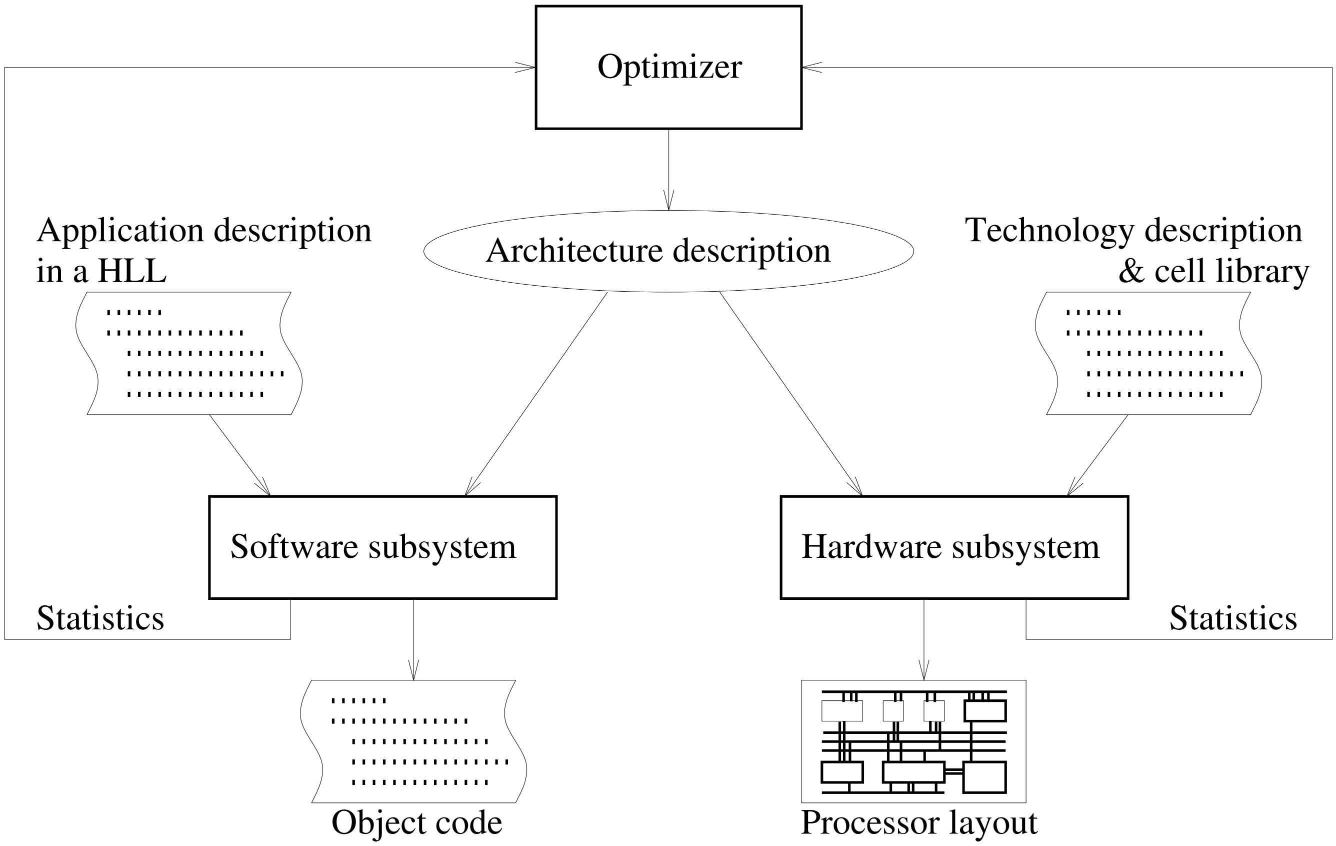


Figure 1.2: MOVE development framework overview.

1.4 Proposed solution

A way must be found to speed up the evaluation of the myriad of possible MOVE configurations for a given set of requirements. The solution elaborated in this report falls apart in two parts.

The first part is the need for a way to efficiently (quickly) transform a MOVE configuration parameter set into a design ready for fabrication (and evaluation). In ”A MOVE Processor Generator” [Lam93], this problem is tackled using the ASA silicon compiler [Sag93]. In this report the development of a new processor generator is described. It has been implemented using design tool packages which are currently considered to be the world's industrial standard. The choice for these tools was primarily governed by the need for a processor generator that could produce better results (faster designs) than the existing one. The characteristics of the designs generated with the two processor generators will be compared.

The second part of the solution lies in speeding up the characterisation of large series of configurations. Until now, it was necessary to go through the entire design trajectory for each processor configuration in order to obtain information about these characteris-

tics. A more efficient approach, based on the reusability of characterisation data, is also discussed in this report.
It should be noted that the main focus of this project was the development of a new processor generator. It's for this reason that the second part of the solution, more efficient characterisation of series of MOVE configurations, will only be addressed in a theoretical fashion.

1.5 Report structure overview

In chapter 2, the problem of generating processor designs will be elaborated on. Chapter 3 will focus on characterising MOVE processor configurations. In chapter 4, an overview is given of the proposed solution of the two problems sketched in chapter 2 and 3. Chapter 5 describes the new processor generator, in which the proposed improvements described in chapter 3 have been implemented. Chapter 6 contains the design proposition for a modeller tool, implementing the observations made in chapter 3. In chapter 7, some experimental results of the new processor generator are given, as well as a comparison with earlier MOVE designs. Chapter 8 contains the conclusions.
Appendix A contains descriptions of the functional units implemented in the new MOVE processor generator. In appendix B, the supported architecture parameters are described. Appendix C contains the parametrised VHDL description files of the MOVE processor generator.

Designing and implementing MOVE processors, current state of affairs

2

This chapter will focus on the ways in which MOVE designs can be generated. In the first section, manual processor design will be discussed. The second section describes automatic processor synthesis, using the ASA MOVE Processor Generator. The final section lists how the automatic processor synthesis method can be improved further.

2.1 Manual design and realisation

The first MOVE processor developed at the Computer Architecture and Digital Technology laboratory was the MOVE32INT [Cor93], [CvdA93]. It was realised using a Sea of Gates (SoG) process [Str91] with a gate-length of 1.6 μm. The entire processor was designed full-custom, using the Nelsis design tools [GS92]. Though the use of SoG instead of full-custom design reduces the design and layout effort considerably, this method still is very labour-intensive.

The performance of this design, as opposed to automatically synthesised processors, is excellent. However, reconfiguring the design (adding a MOVE-bus, for example) is problematic, as the manually placed-and-routed design is completely fixed. This means that for almost every new configuration, an entirely new design must be made, from the beginning (fig. 2.1).

From this, it can be concluded that manual design, though it may deliver good-performance designs, is not a very practical alternative when a lot of different design

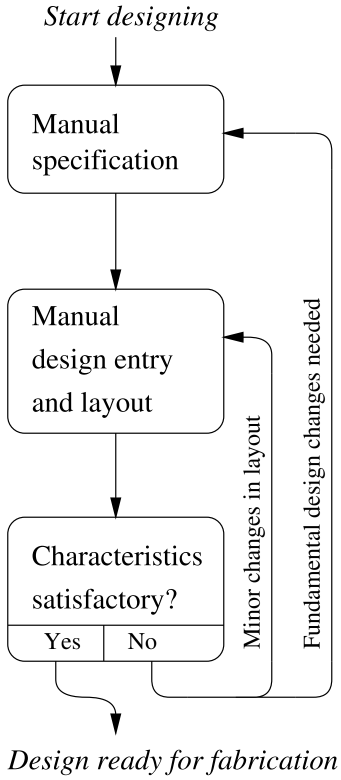


Figure 2.1: Manual design procedure.

alternatives need to be tried out.

2.2 Automatic synthesis

The first MOVE Processor Generator [Lam93] uses Sagantec's ASA silicon compiler [Sag93] to do automatic logic synthesis and placement and routing. From a parametrised description template (written in SID, ASA's hardware description language, and augmented with various preprocessing commands) and a file with architecture parameter declarations, it generates a full description (in synthesisable SID) of the processor to be implemented. This description is then synthesised and laid-out with the ASA tools.

An overview of the design cycle is given in figure 2.2. First, a synthesisable description of the processor is generated, usually with the help of a preprocessor. Next, the synthesis tool performs high-level and logic synthesis (or just logic synthesis, depending on the description style). A netlist results, from which intermediate characterisation results can be derived. If the outcome of this characterisation is unsatisfactory, the processor configuration can be altered (effectively restarting the design cycle), or the design can be re-synthesised (with new optimisation constraints). If the gate-level characterisation results meet the constraints, then the design can be transformed into a chip layout, using placement and routing tools (layout synthesis). This layout can also be characterised, resulting in more accurate values. If these values are also satisfactory, then the design is ready for fabrications. If not, then the layout can be re-generated using a different placement and routing strategy, or the designer can decide to start all over again by changing the processor configuration.

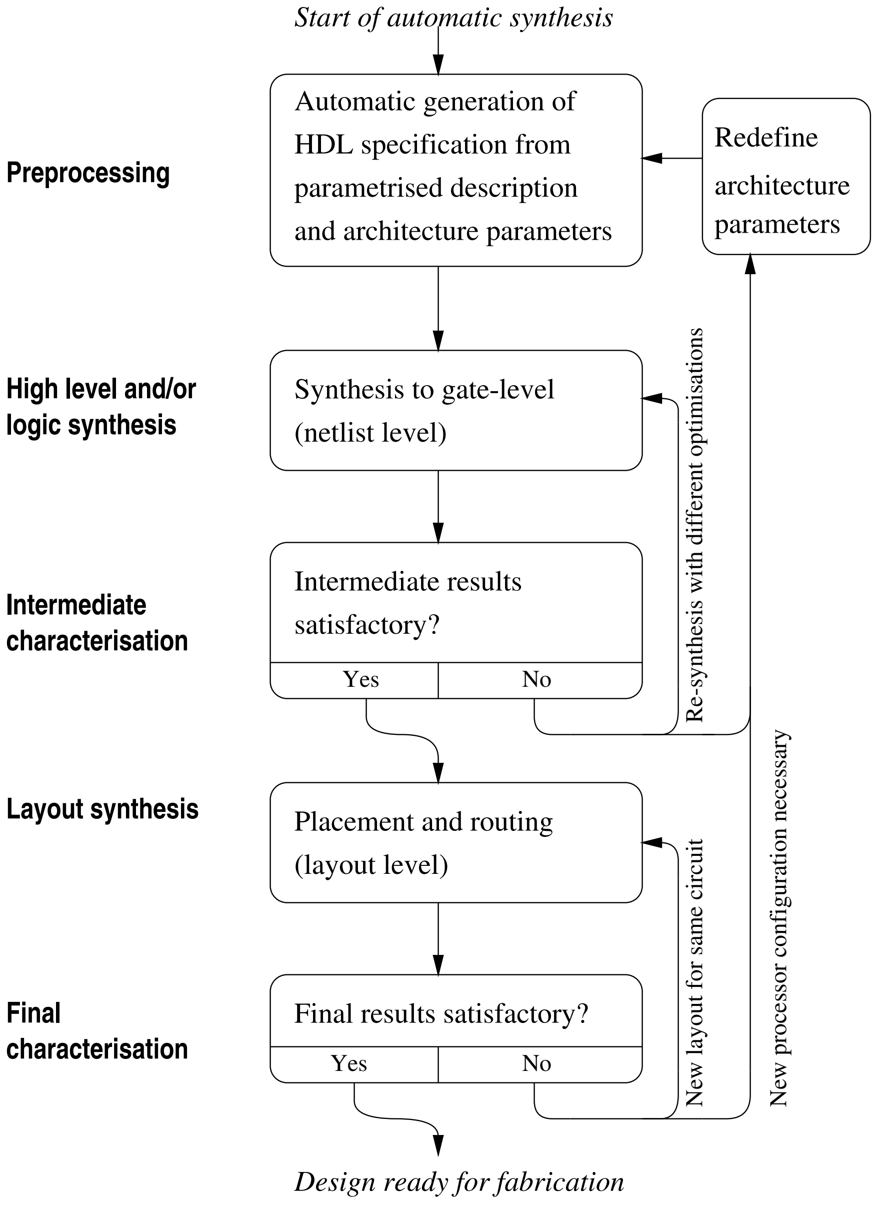


Figure 2.2: Automatic processor synthesis procedure.

A MOVE32INT-compatible processor called the MOVEASAINT [Lam93] has been synthesised with the Processor Generator. It was implemented using the ES2/CMS1M0 and ES2/CMS1M5 processes (1.0 and 1.5 μm channel width CMOS, respectively). The area and timing characteristics, along with those of the SoG version of MOVE32INT, are shown in table 2.1.

Processor	**Process**	**Area**	**Dimensions**	**Max. delay**	**Clock freq.**
MOVE32INT	SoG (1.6 μm)	1 cm^2	1×1 cm	12.5 ns	80 MHz
MOVEASAINT	ES2/CMS1M5	2.61 cm^2	1.42×1.84 cm		$\sim$27 MHz
MOVEASAINT	ES2/CMS1M0	1.07 cm^2	0.90×1.19 cm	23.3 ns	43 MHz

Table 2.1: Characteristics of existing MOVE designs.

Though the performance of the synthesised design may not be as good as when the design were made by hand, the advantage lies clearly in the much shorter design cycle time. It should be noted that the version of the MOVEASAINT architecture that is used in this report was by no means the final one, a lot of improvements have been made since. This version is used as a reference because it is well-documented.
It is also a lot easier to re-configure the design. Different configurations can be implemented by simply modifying the parameter declarations file and re-synthesising the resulting HDL description.

2.3 Room for improvement

In the previous two sections, it became clear that there are both advantages and disadvantages to automatic processor generation (as opposed to full-custom design).
Advantages include:

- *Rapid development.* The design time with automated processor generation is only a fraction of the time full-custom designing takes.
- *Scalability.* Various MOVE configurations can be implemented by merely altering the architecture parameters.
- *Architectural flexibility.* Implementing new architectural features is easy, since all that needs to be changed is the parametrised HDL description.
- *Fabrication process flexibility.* Designs can be easily re-targeted to different fabrication processes.

The main disadvantage is the quality of the automatically synthesised design in terms of area cost and timing performance, as was shown in table 2.1.
The advantages of automatic processor synthesis are clearly beneficial to the purpose of the MOVE framework: rapid and flexible processor design. When implementing a new processor generator, it will be important to retain these advantages while simultaneously trying to improve the performance of the generated processors.

One way of trying to improve the quality of the results could be to use more powerful software tools than ASA. It would be interesting to see whether a MOVE processor generator that uses an industry standard HDL and synthesis tool would deliver better designs. A comparison could be made between how the different processor generator perform the same task.
The two most widely accepted hardware description languages are VHDL and Verilog. VHDL was chosen because of previous experiences and the availability of several simulators on which the processor description could be tested.
The Electronic Design Automation tools that are today's industry standard would be the Synopsys Design Analyzer and Design Compiler as synthesis package and the Cadence Design Framework II as layout tool. These have been selected solely on reputation and user base, since it is not possible to determine beforehand which tools will perform best at this particular task. Other candidates would have included the tools from Mentor Graphics and Compass, though previous experience has shown the latter to perform less well at generating MOVE processors than ASA [Hof94].
During the development of the first MOVE Processor Generator, several limitations of Sagantec's hardware description language SID and its language compiler surfaced [Lam93]. These limitations have been partly overcome with elaborate preprocessing of the parametrised description files. It would be desirable to have simpler, and less preprocessing. This would make the input files more readable and understandable, thus making it easier to modify the processor description if functionality must be added or changed.

3 Characterising MOVE configurations, current state of affairs

During or after designing a processor, it's necessary to characterise the processor design in terms of chip area used, the minimum cycle time (or maximum system clock frequency) and/or power consumption. These values are needed to calculate the cost/performance ratios of the different processor configurations that are to be compared. For the purposes of the MOVE framework, where a large amount (hundreds or thousands) of processor configurations need to be evaluated, it is important that the characterisation is done in as little time as possible.
In the first section of this chapter, the conventional way of characterisation is described. The second section focusses on the MOVE*mate* tool, which was implemented to exploit the similarities between different processor configurations. The last section gives an idea on how the characterisation process can be optimised even further.

3.1 Evaluating after a complete design cycle

In figure 3.1, the trajectory for a typical synthesis run is shown. First, the behavioural level HDL description is processed by a tool that does high-level synthesis, followed by logic-level synthesis. The result is a gate-level description of the design. This description is then transformed into an actual chip layout by a placement and routing (layout synthesis) tool.
Usually, a design is characterised at layout-level, only after the complete design trajec-

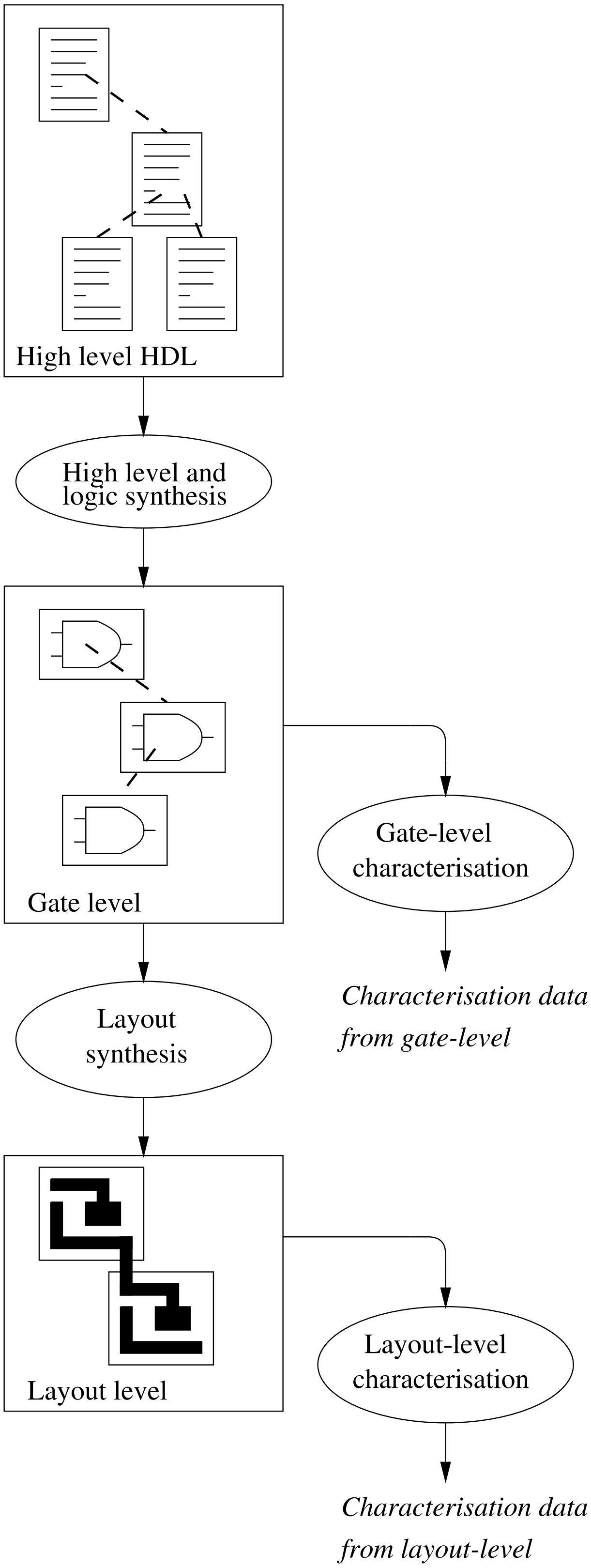


Figure 3.1: Characterisation at gate and layout level.

tory has been completed (fig. 3.1). Characterisaton (e.g. timing analysis) is done by extracting physical characteristics like resistances and capacitances from the design at layout level. From this data, the design's characteristics are calculated. Most computer aided engineering tool packages can do this automatically and the results can be very accurate, depending on the detail of the extracted data. The time this approach takes is considerable, though.

For the purpose of the MOVE framework however, which is characterising many similar but different processor configurations, layout-level extraction takes far too much time. It means that for every new configuration, a completely new design at layout level must be generated and then characterised. Even when this design is synthesised automatically, the time cost is considerable, as "generating layout may take days" [Lam93].

Intermediate characterisation at gate-level is a faster, but less accurate way of evaluating a design. It disregards the effects of placement and routing which are only visible at layout-level. Even though the evaluation cycle may be shorter this way (no placement and routing is done, very time-consuming tasks), it is still necessary to synthesise the entire design each cycle. The next section discusses the MOVE*mate* modeller, which was conceived as a solution for this problem.

3.2 MOVE*mate* modeller

Until now, the different MOVE processor configurations have been treated as completely separate designs. In practice, however, they turn out to often have a lot in common. A MOVE processor can be divided into several parts whose internal structure depends on only a subset of the architecture parameters, rarely on all. This means that configurations, for which some parameters have the same values, result in designs in which the parts depending on these parameters are the same. When characterising these designs, it is not necessary to analyse these parts again and again for each design. Instead, the data obtained from a previous design can be used.

This reusability is exploited in the MOVE*mate* modeller [vdL95]. What it does, basically, is store the information obtained from each cell in a database, indexed by architecture parameter value. When architecture parameters are changed, the MOVE*mate* searches the database for characterisation information for all cells in the processor, using an internal model of the processor cell hierarchy (figure 3.2). Only the cells for which no information is found are evaluated with the processor generator (in this case, the ASA version).

This results in a characterisation process which, at first, takes as much time as characterisation with the processor generator. After processing several configurations, however, the database becomes fuller and the chances on finding the information in the database increase. This could result in a much shorter characterisation time for the series of configurations under consideration, especially when they have a lot in common, parameterwise.

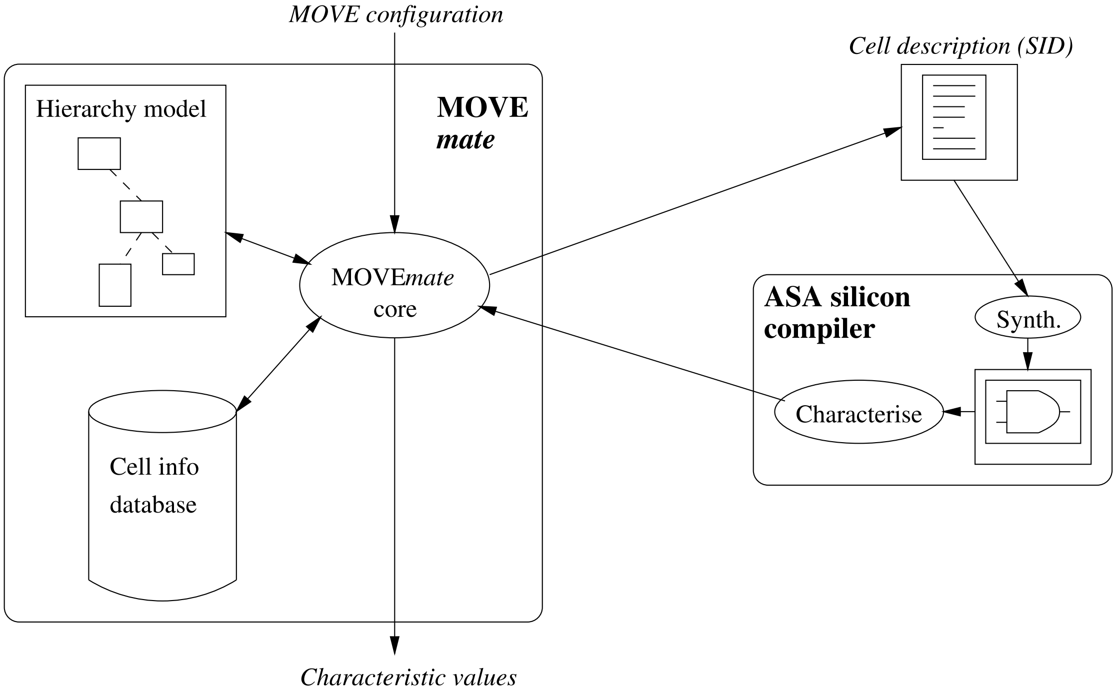


Figure 3.2: MOVE*mate* structure and interface to ASA.

3.3 Room for improvement

In this section, some suggestions are made as to how the modeller could be improved to make for an even faster characterisation process. Some of these can also be found in [vdL95].

- The current MOVE*mate* modeller only looks in the database for exact parameter value matches. It is conceivable, however, that a mathematical or statistical relation can be found between the values of a cell's characteristics and the values of the parameters those characteristics depends on. If that's the case, then approximations may be made for these characteristics, removing the need to use the (much slower) processor generator characterisation option for the cell under consideration.

- If the processor generator must be used for characterisation, then information about area, timing and power can be obtained on two levels. The information from layout-level is the most accurate of the two, but also takes the longest to obtain since the calculations involved are much more complex. The information obtained at gate-level is somewhat less accurate since it disregards placement and routing information, but it's calculated much faster. Now, it may be so that after several characterisation iterations, the gate-level information turns out to deviate from the layout-level information with, for instance, a fixed percentage. If that's the case, then the gate-level information can be used to approximate the layout-level values, resulting in more accurate values in only slightly more time than the gate-level analysis costs.

- The MOVE*mate* tool has been tailor-made for use with the ASA version of the MOVE processor generator. A modeller may be more useful in the MOVE framework if it can cooperate with several different silicon compilers. A compiler-independent modeller, interacting with the compiler through a separate interface, would make the modeller easier to adopt to new (or updated versions of) silicon compilers; all it would take is a new interface.

- To compensate for the absence of placement and routing information, the MOVE*mate* modifies the design area with a global wiring factor, just as ASA does. It may be more useful to use a local wiring factor, which could be obtained in two ways. The first way is to extract area information for the cell on both gate and layout level. The layout level information divided by that of the gate level would yield the wiring factor for the cell. The second (faster) way would be to make an approximation, based on the number of nets, gates and pins of the cell. This approximation could be given beforehand, explicitly, or it could be calculated by the modeller.

- The timing model of the MOVE*mate* is very similar to that which ASA uses. A more generic model may be appropriate to facilitate cooperation with other silicon compilers. The translation of the data from the compiler's format to that of the modeller should be taken care of by the interface part, to make the rest of the modeller independent of the actual processor generator that is used.

4 Overview and requirements of the new MOVE processor design and evaluation system

This chapter will feature an overview of the MOVE processor design and evaluation system to be developed. From the previous chapters, it becomes clear that the system should consist of two independent but interacting parts: the processor generator and the modeller (fig. 4.1). The interaction is realised through an autonomous part of the modeller: a separate interface program. All that will be needed to enable cooperation between future modeller and processor generator implementations are new interfaces.

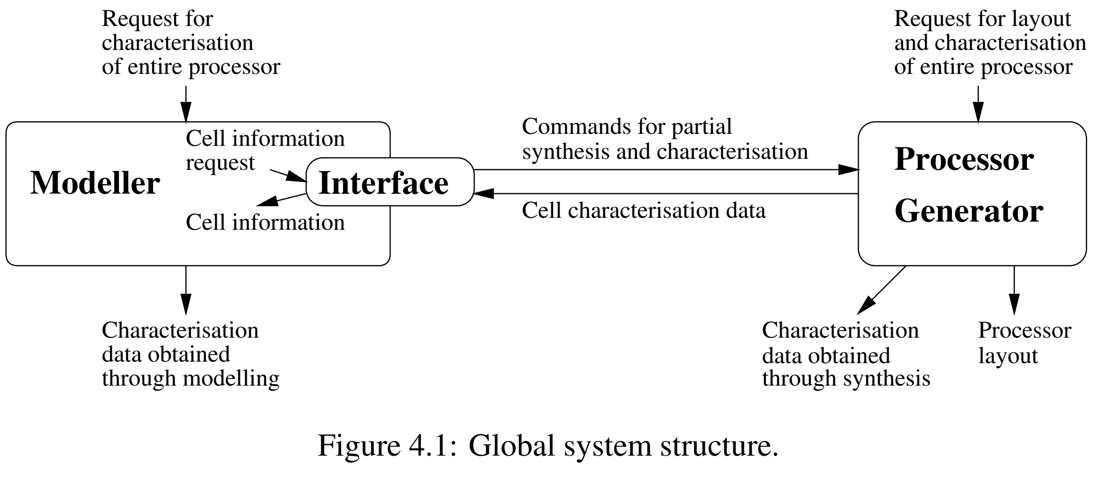


Figure 4.1: Global system structure.

In the first section of this chapter, the requirements for the new MOVE processor generator will be stated. The observations from chapter two will be taken into account. The second section contains the requirements for the new modeller, derived from the observations in chapter three. How the processor generator and the modeller should interact is described in section three.

4.1 Requirements for the MOVE processor generator

The primary, functional requirements for the new processor generator are the same as those for the ASA version:

- The processor generator must be able to generate processor with the MOVE architecture.
- The processors that are generated must be compatible with the MOVEASAINT architecture, to allow comparison between the two processor generators.
- The processors must be scalable by means of user-defined architecture parameters. The scaling must be automatic.
- The architecture must be easy to modify, new architectural features must be easy to implement.

The primary function of the new MOVE processor generator is the same as that of the ASA processor generator: speeding up the MOVE processor development process. This is done by taking as much design effort off the designer's hands as possible. The designer should not be bothered with work that is the same for each MOVE processor design, this should be automated.
The secondary function of the new processor generator follows from chapter three: to synthesise subcells of the MOVE processor hierarchy and characterise them in terms of used chip area, critical delays and/or power consumption.
Possibilities for automation lie in every step of the design process:

- *Design specification:* A very effective [Lam93] way to automate this is through the use of a standardised processor description in a hardware description language (HDL). This description must be modified according to the configuration's parameter values, resulting in a complete HDL specification of the MOVE processor instance. From the secondary requirement it follows that it must be possible to generate HDL descriptions of subcells, which can be synthesised independently. It is desirable that the achitecture parameter dependencies of those subcells can be extracted from their descriptions. It is also desirable that the HDL description has wide industrial support, for portability reasons. It is conceivable that, in the future, another processor generator will be built around another silicon compiler, but with (preferably) the same input HDL description.

- *Design entry/optimisation:* The step from behavioural HDL specification to gate-level (netlist) can be automated through the use of existing synthesis software [Mic94]. These programs are able to read high level HDL descriptions and turn them into netlists. The quality of these netlists in terms of used chip area, timing characteristics and power consumption depends strongly on the amount of effort (processor time) put into optimising the synthesised design. The designer must be allowed to control this effort. One will likely want to spend less effort on synthesising one of many (sub)designs just for evaluation purposes than when synthesising a design 'for real', i.e. for production.

- *Layout generation:* Placement and routing of the design can also be done automatically. Though this is still much faster than manual placement and routing, the time needed exceeds the time for high-level and logic-level synthesis by an order of magnitude. It is for this reason that characterisation at gate-level is more suitable to the needs of the modeller than layout-level analysis. If, however, exact figures are needed, then characterisation at layout-level should still be possible.

4.2 Requirements for the modeller

The modeller is to be used to further speed up the secondary function of the processor generator, which is characterising MOVE processor configurations in terms of occupied chip area, maximum clock frequency and/or power consumption. In order to capitalise on the observations made in chapter 3, the modeller should meet the requirements in the following paragraphs.

- *Calculate characterisation data:* A MOVE processor can be seen as a hierarchical structure of cells. The modeller should be able to gather information for all cells in the hierarchy and calculate the characteristic values of the entire design. To this end, it is necessary to create an internal representation of the cell hierarchy to use as a framework from which to work.

- *Model the hierarchy:* All cells in the hierarchical structure that the MOVE processor is should be separately accessible. This is necessary to enable the modeller to make, for each cell, a choice on how to obtain the characterisation data. Possible sources include: previously obtained (exact) values that have been stored in a database, approximations based on other, closely related entries in the database and, failing that, exact data obtained from the processor generator.

- *Store characterisation data* per cell, along with the parameter values the data depends on. All characterisation data (obtained via the processor generator) should be kept in a database to allow reuse. This data should be in a standard, silicon compiler independent format. All data for a certain cell should be accessible at all times, not just the data corresponding to the actual parameter values at that time. This is needed to be able to supply the approximator part of the modeller with enough data to base its estimate on. Note that it is important that only exact values are stored in the database, not values coming from the approximator. Otherwise, approximations would be based on previous approximations, which would lead to increasing (and uncontrollable) inaccuracies.

- *Approximation:* A separate part of the modeller should concern itself with trying to find values that are not in the database, but for which, on the basis of the other entries, an approximation can be found. Along with the approximation, an accuracy estimate should be given on the basis of which the modeller can decide whether to use the value or not. The approximator should be implemented separately from the main part of the modeller because it may be desirable to try several different approximators in combination with the same modeller.

- *Obtaining data from the processor generator:* If no data is available from previous designs or if approximation results are unsatisfactory, then the only course left is to use the processor generator to obtain the data. This should be done with a separate interface, allowing cooperation with different processor generators through different interfaces. The requirements of these interfaces are discussed in the next section.

4.3 The interface between processor generator and modeller

The function of the interface part is to translate information between the modeller and the processor generator. Standardised requests for information from the modeller must be translated into an input the processor generator understands. The requests from the modeller will consist of the name of the cell that must be characterised, as well as relevant architecture parameters for the cell. This information should be used to direct the processor generator to synthesise the cell and determine its characteristics.
Once the cell has been synthesised with the processor generator, all of the cell's characteristics (area, timing, power) are available. It makes sense to extract them all and store them all in the modeller's database, even if only one of them is needed at the moment. Even if not all the information is wanted now, it may be later. It should be noted that even if some of the exact characteristics will never be needed directly, they may still be useful to the approximator.
Before the information from the processor generator can be stored in the modeller's database, however, it should be converted to the standard modeller format.

Another task of the interface might be to convert the architectural implementation data found in the HDL source files into the Implementation Description File (IDF), which the modeller takes as input. Whether this part should be actually implemented as a program, or whether the task can be performed manually, depends on how often architectural modifications are expected to take place.

5 Implementation of the new MOVE processor generator

In this chapter, the actual implementation of the second MOVE processor generator will be described. The processor generator can be divided in several parts (fig. 5.1). The first part, the preprocessor, concerns itself with generating the actual processor description in VHDL. This part is described in section 5.1. The second part, the high level synthesis tool, is described in section 5.2. It takes the high level VHDL description as input and writes it out into an EDIF netlist. The software that generates actual chip layout from the netlist (layout synthesis) is described in section 5.3. Finally, in section 5.4 the differences with the MOVEASAINT implementation [Lam93] are discussed.

5.1 Generating the HDL processor description

The first stage of the processor generator consists of generating a complete description of the MOVE configuration to be implemented, in a format that the next stage, the high level and logic synthesis software, accepts. This description is generated in a way that is similar to the approach taken in the first processor generator: a standard, parametrised HDL description is modified with a preprocessor, resulting in a description which described the configuration for the current architecture parameter values. The reason for this approach will become clear in the next paragraphs.
In paragraph 5.1.1, the language used for the description is discussed. The second paragraph, 5.1.2, contains information about the structure of the HDL description. In paragraph 5.1.3, the necessary preprocessing is described.

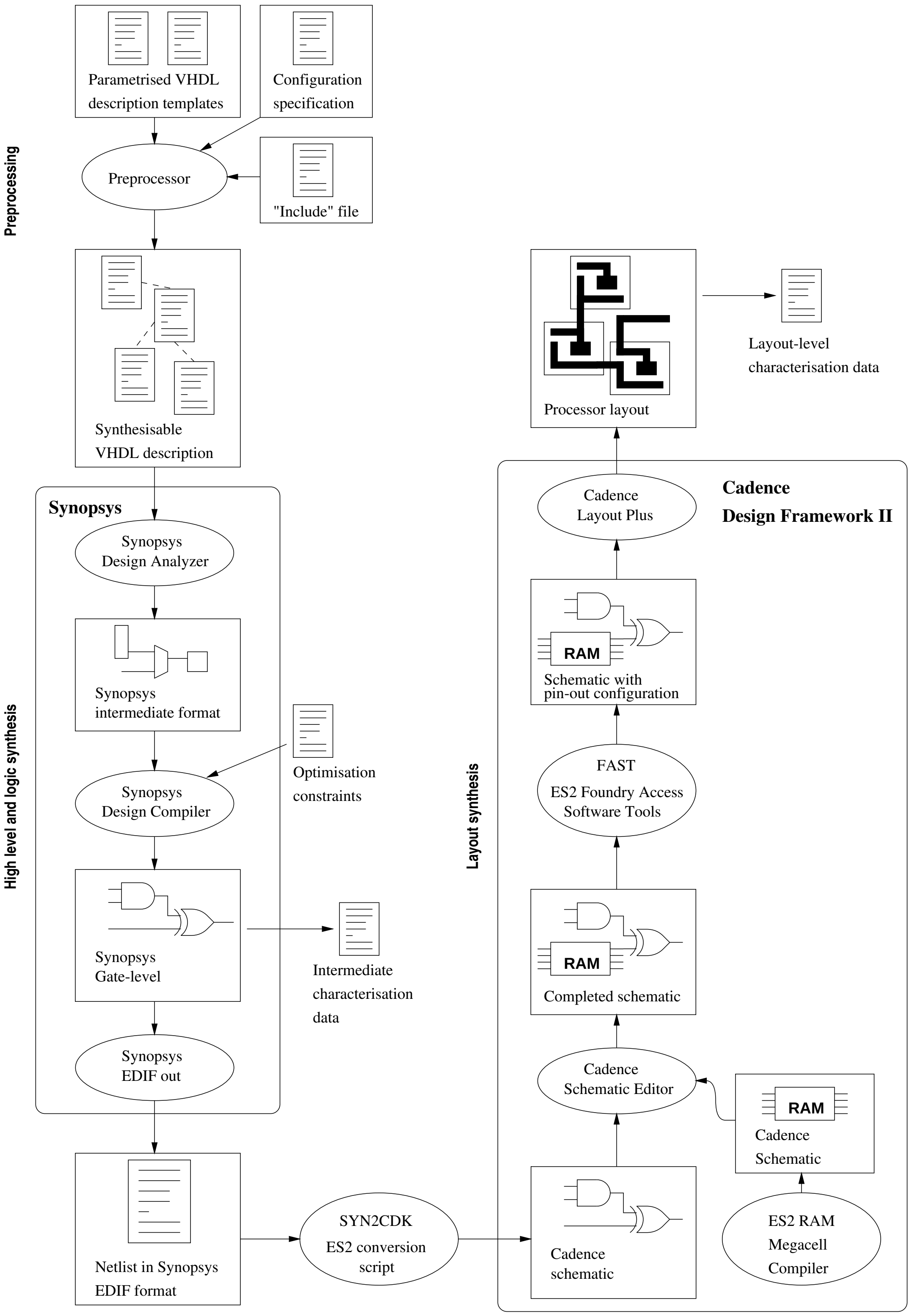


Figure 5.1: The implementation of the new Processor Generator.

5.1.1 Choosing the Hardware Description Language

From the portability requirement, mentioned in chapters 2 and 4, it follows that the description should be implemented in a widely accepted Hardware Description Language. Currently, the HDL accepted by most commercially available tools is VHDL (*V*ery High Speed Integrated Circuit *H*ardware *D*escription *L*anguage).
Originally conceived as a documentation language for the US Department of Defense, VHDL was later (1987) adopted by the IEEE as the standard hardware description language (IEEE standard 1076-1987, [IEE87]). VHDL is now widely used as a means to specify designs in the development phase [HS91], [BFMR92], [Mic94].
A drawback of VHDL's origins as a documentation and modelling language [Arm88] is that VHDL code tends to be structured quite rigidly. It's often not possible to write generic constructs whose final form is determined by parameters, global or local. For instance, it's not possible to let the existence of a certain port depend on a parameter value, only its size. It's because of limitations like these that preprocessing was needed. This is described further in paragraph /refsubsec:preprocessing. Another drawback of its origins is that VHDL descriptions tend to be fairly verbose; the language was "developed for readability rather than writability" [BFMR92]. This has some consequences for the structure of the parametrised VHDL description and the ease with which it can be modified. This will be discussed further in the next paragraph.

5.1.2 Structure of the parametrised VHDL description

In this subsection, the structure of the parametrised VHDL description is discussed. First, however, a very brief overview of VHDL will be given. A more elaborate description of VHDL can be found in [Per90], [BFMR92], [Nav92].
The two most important design objects in any VHDL description are **entities** and **architectures**. Entities describe a design in terms of input and output ports. Architectures are coupled to entities and describe the inner workings of those entities. There are several coding styles that can be used when writing architectures: behavioural (sometimes called functional), dataflow and structural [BFMR92].

- Behavioural code is the highest level, this means the most abstract, but also the most flexible code. Descriptions written on this level read like normal (non-HDL) programs and are thus easier to understand and easier to modify.
- Dataflow is an intermediate level where all functionality is described in terms of simple logical functions and signal assignments.
- Structural level describes actual gates and their interconnection. Descriptions on this level are effectively netlists if the gates they use model the target process technology's gates.

The parametrised VHDL description of the MOVE processor has been kept on as high a level as possible, because high level code has several advantages over lower levels:

- It is easier to understand.
- It takes less effort to change the functionality of the architecture, because it is easier to pinpoint the code that must be changed.
- New functionality is easily implemented because high level code is easier to write.
- The synthesis software has more freedom to decide on the eventual (logic level) implementation. Structural level dictates the structure of the design, leaving little room for optimisation.

Note that, at first sight, the last item appears to have a downside: the synthesis software may have more freedom, but this also means that the designer has less control over the eventual implementation. This may be so, but the high level synthesis software's implementation decisions are governed by optimisation constraints, which are set by the designer.
The hierarchical structure of the MOVE architecture is clearly reflected in the VHDL architectures. Each non-leaf cell contains a number of **components**. These are design objects that represent subdesigns: instantiations of entity/architecture pairs.
The different levels in the hierarchy are, from top to bottom:

- *Global interconnection:* The MOVE buses are embedded in this level. It does not contain any logic, only interconnections between functional unit components.
- *FU level:* There is an FU entity/architecture pair for each functional unit type, containing the in/output sockets and FU internals. The functional units that were added since the MOVEASAINT architecture implementation are described in appendix A. For descriptions of the other FUs, the reader is referred to [Lam93].
- *Sockets/internals:* Sockets and FU internals lie on the same level. The 'internals' entity/architecture pairs interconnect the stage control components and the combinational FU logic.
- *Stagecontrol/combinational logic:* these are on the lowest level and do not contain any subcells.

Note that the instruction fetch FU has a slightly different structure because of its greater complexity.
The VHDL description does not yet contain the bonding pads and the instruction fetch cache RAM. These are not added until the layout synthesis phase. It would not make sense to use high level and logic synthesis to generate an entire RAM bank from a VHDL description. The amount of CPU time and memory needed would be excessive, almost always exceeding the available resources. Besides, the possibilities for optimising such a regular structure as a RAM are negligible. The instruction cache RAMs are therefore represented by empty entities with only the input and output ports of the actual RAM

cells that will be substituted later. The limitations in size, wordlength and wordcount of the RAM cells for the desired implementation technology can be specified in the parameter file. The preprocessor then makes sure that the dummy RAM cells have the proper dimensions: it must be possible after all to substitute actual RAM cells with the same dimensions.
A consequence of the verbosity [BFMR92] of VHDL is that many declarations must be done twice or more (e.g. both at entity and component levels). This would make implementing changes in the standardised description tedious and repetitive work. To eliminate this as much as possible, an approach using so-called 'packages' has been chosen.

An example of the use of packages: Usually, a component can only be instantiated if it has been declared inside the architecture in which it is used. So, if the entity 'add_full' should be used as a component in the architecture 'arch' of the entity 'add_4', this would be the only correct syntax:

```
achitecture arch of add_4 is

 component add_full                -- Component declaration
   port(a, b, c : in std_logic,    -- std_logic is the standard IEEE
        y       : out std_logic);  -- defined data type.
 end component;

begin
  adder1:add_full port map(a1, b1, b1, y1);
     -- Component instantiation
end;
```

Since it is only visible within the architecture where it is made, the component declaration should be repeated in each architecture where the entity add_full must be instantiated. That is, unless the component declaration were to be done in a package, then only the package would have to be made visible with the 'use' statement:

```
package add_pack is                -- Package declaration
 component add_full
   port(a, b, c : in std_logic,
        y       : out std_logic);
 end component;
end package;

use work.add_pack.all;
     -- 'use' statement, the contents of the design object
     -- 'add_pack' is now visible from within the following
     -- architecture.
```

```
achitecture arch of add_4 is
begin
  adder1:add_full port map(a1, b1, b1, y1);
     -- Component instantiation
end;
```

This may still seem fairly verbose, but the advantage lies in the fact that now the component declaration is done only once, in the package body. Each time it is instantiated, all that is needed is the 'use' statement. This lessens the chance for making typing errors while writing the code, and it also makes for a nice reduction in code size. The advantage of the latter part is not only a more readable VHDL description, less code also means less compile time for the high-level synthesiser.
The VHDL description for the MOVE architecture includes separate packages for socket, combinational FU logic, FU internal and FU component declarations.

5.1.3 Preprocessing

Though VHDL does support so-called generic parameters, its syntax does not allow parametrisation to the extent which is needed here. For instance: it is possible to specify the width of an input or ouput port vector with a generic parameter, but it is not possible to make the existence of this port dependent on a parameter value. This is but one of the reasons why the VHDL parameter syntax is far too limited. Therfore, a preprocessing phase had to be introduced, during which the architecture parameter values are used to create the VHDL description for the corresponding MOVE processor configuration. How this is done is shown in figure 5.2.
All preprocessing (except for the SED-based textual beautification at the end) is done with one preprocessor: m4 [M4,92]. This preprocessor can be found on most UNIX systems. Compared to the preprocessing used in the previous processor generator [Lam93], the number of passes has been greatly reduced, from 15 to 2 (or 3, if the SED phase is counted as a pass. It is possible to skip this phase, however: it was merely added to make the VHDL description more readable to the human eye).
First, m4 macros are defined using the user defined and derived parameters. This is done with the `params` meta-source file, which uses three source files of its own:

- The `params_cons` file contains implementation-defined constants such as functional unit delays. This file should only be changed if the architecture (the parametrised VHDL files) are changed.
- The `params_user` file contains the user-defined architecture parameters such as buswidth, number of MOVE buses, number of different functional units, etc. The structure of this file was kept nearly identical to the 'parmdecl' file of the ASA processor generator. This was done to make writing the interfaces (section 4.3) between the modeller and the processor generators easier.

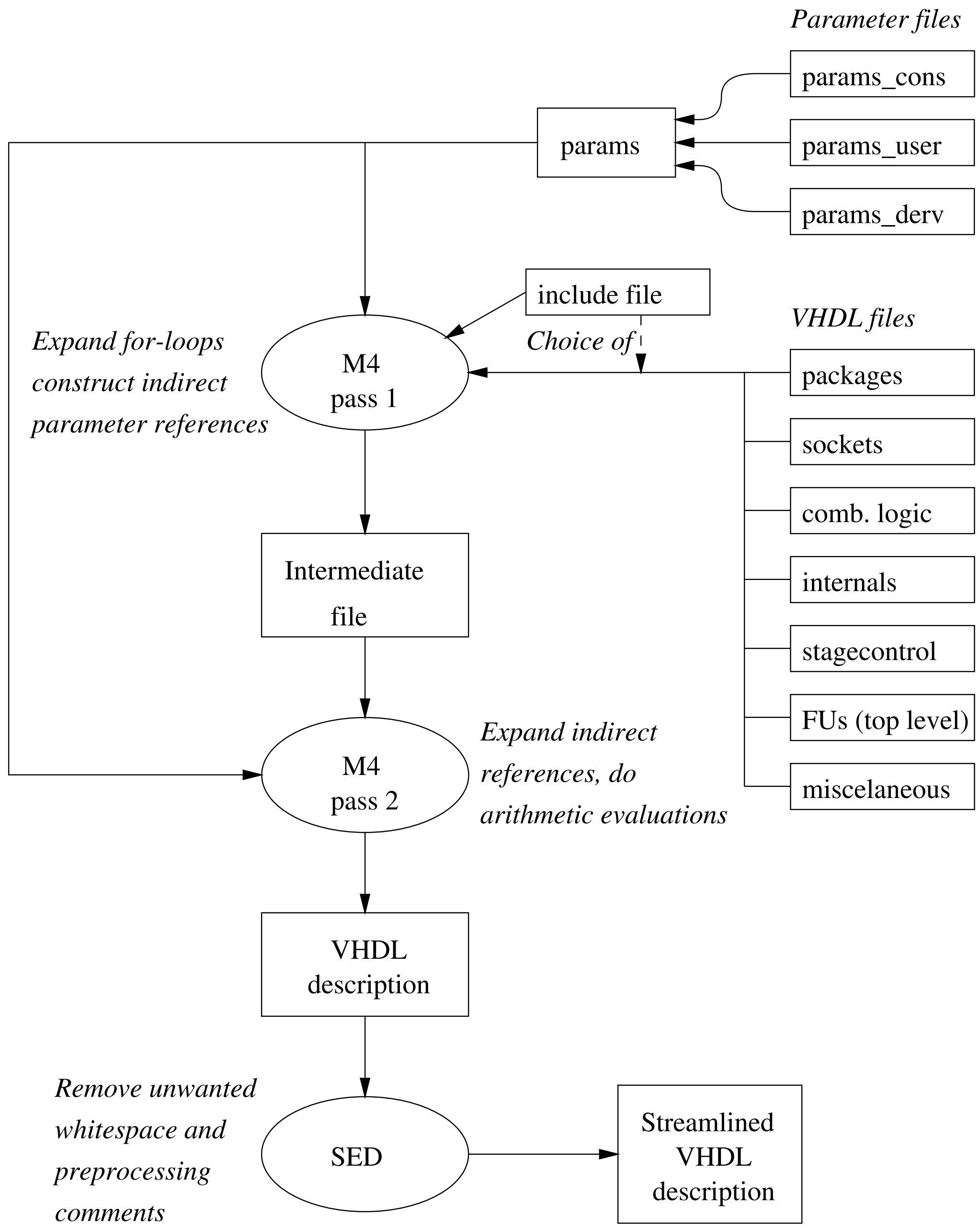


Figure 5.2: Preprocessing of the VHDL description.

- The `params_derv` file is used to calculate derived parameter values, that is parameters whose values are derived from the values of the user-defined parameters and implementation-defined constants. These are mostly addresses (IDs) for the input and output sockets, the number of local lock request lines for each FU, exception request numbers, etc. The derived parameters are mainly used to define the connections of the transport network to the FUs. This is done exactly the same way as in the previous Processor Generator.

During pass 1, it is decided (by means of an 'include' file) which design entity/architecture pairs (cells) will be included in the output VHDL description. This is done to enable synthesis of partial designs, as stated in the requirements (section 4.1).
The included files are scanned for macros when they are read. These macros are the parameters defined with the 'params' file or command macros used to do evaluations with the parameter values (such as the 'for' loop macro). In the first pass, for loops are expanded and parameter macro names are substituted with their values.
It is necessary to do a second pass because of indirect parameters. These are parameter macros whose names are composed of a static (literal) part and a part that depends on the value of other macros.

An example of an indirect parameter name:

CINTFU''funr''_TBUS''busnr

In this example, "CINTFU" and "_TBUS" are the literal parts and "funr" and "busnr" are macro names. During the first pass, the values of the macros funr and busnr (which may be, for instance, for-loop counter variables) are substituted. This results in an intermediate result of, for this example:

CINTFU''0''_TBUS''3

In the second pass, the quotes ('') are eliminated so that the resulting name, CINTFU0_TBUS3, is substituted with the value that was assigned to it by the user, in this case 0 or 1. (see the `params_user` file for more about this).

After the second pass, all parameter macros have been expanded to create a syntactically correct, synthesisable processor description.
In order to eliminate all unwanted whitespace and preprocessing commentary that is a side-effect of the two m4 preprocessing stages, the VHDL file can be put through the stream text editor sed. This is merely a beautification pass and not necessary from a syntactical point of view, however.
The resulting (beautified or not) VHDL file can then be fed to the silicon compiler, the second stage of the MOVE processor generator.

5.2 The Synopsys high-level and logic synthesis software

For the high level and logic synthesis part of the new processor generator, the Synopsys Design Analyzer [Syn94a] and Design Compiler [Syn94b] were chosen. The motivation behind this was the demand for a processor generator which used an 'industrial strength' tool that would be more widely accepted than ASA, and which could read VHDL as input instead of SID. (Note: the latest version of ASA (v6.2), which was released about halfway during this project, also accepts VHDL. It would be interesting to see how the two compilers perform with the same input files.)

The Design Analyzer is the part that parses the VHDL input file and stores it into an internal database, in a Synopsys-specific format. Only global synthesis is done at this stage, the design is rewritten in terms of generic logic/sequential cells and more complex structures such as multiplexers and adders are represented by Synopsys' own synthetic cells called 'DesignWare' [Syn94c].

At this point, optimisation constraints must be given (such as clocks that must be assigned to the appropriate ports) and a target technology and operating conditions must be specified. After this, the designer must decide how much compiler effort must be spent on this design and what optimisation strategies must be taken: logic structuring and/or flattening, boolean optimisation, minimum path prioritising, boundary optimisation (allows the compiler to look across cell boundaries), etc. The reader is invited to study the Synopsys manuals ([Syn94a], [Syn94b] and [Syn94c]) for details on this.

After this, the Design Compiler takes over. At this stage, actual gate level mapping and optimisation is done using the specified technology library. In this project, the European Silicon Structures 0.7 [ES294a] and 1.0 μm [ES294b] processes were used. The recommended strategy is to first compile the entire design using relatively low effort and optimise it for chip area use. If time constraints aren't met after that, a second, high effort compilation pass should be executed, using incremental compilation. This means that the compiler optimises only the design parts in which the time constraints are violated, not wasting a lot of effort on non-critical parts (Note: this option is only available to Synopsys users with the DC-Expert license).

During the compilation stage, no placement and routing information is available for optimisation since all operations take place at gate-level. To compensate for this, Synopsys uses a wire load model, based on the number of connections for a net and the area of the cell in which it is contained. These wire load models are technology dependent, they're defined in the same technology library as the gates the design is mapped to.

If the processor generator is used to characterise only a single cell, a slightly different approach should be used. In this case, compilation speed is important. Since only a subcell is to be evaluated, only the cell itself should be compiled. Its subcells, provided they haven't changed, can be excluded from compilation using the 'don't touch' option. The Design Compiler will then ignore these cells during compilation and optimisation.

When compilation and optimisation are complete, the gate level description will be ready. Data on used chip area, critical path and power consumption can be extracted at this point. Though not as accurate as data extracted at layout level, this information tends to be on the conservative side, to ensure that the design will not suddenly exceed its constraints when taken to layout level.

5.3 The Cadence Design Framework II layout synthesis tool

After the design has been compiled and optimised at gate-level, it must be transported to a program that can finish the design at layout-level. Instruction cache RAMs must be added and placement and routing must be done.

In this project, the Cadence Design Framework II [Cad94] is used for this purpose. The reason for this choice is the availability of the ES2 design kit for both Synopsys [ES293b] and Cadence [ES293a]. These were supposed to ensure a fluid transition from one system to the other. In practice, however, the respective design kits turned out to be outdated and incompatible with each other. A considerable amount of time has gone into readjusting the design environments and obtaining the proper conversion scripts.

The suggested method for transporting designs from Synopsys to Cadence is exporting the design as an EDIF (Electronic Design Interchange Format) description. That EDIF is anything but a standard is illustrated by the need for an elaborate conversion script which translates the Synopsys generated EDIF file into a schematic that is readable by Cadence.

After the design has been imported into Cadence, the dummy RAM cells must be substituted with real RAM cells. The RAMs are created using the separate, foundry supplied Megacell Compiler [ES293a], [ES294c]. These megacells must then be pasted into the design.

The now functionally complete design must then be processed with the Foundry Access Support Tools (FAST) from ES2. Several design checks can be done and the pin configuration is determined. Consult the ES2 FAST manual [ES294d] and the Engineering Application Notes [ES294c] for more information about this.

The resulting schematic can then be fed into the Cadence placement and routing utility, LayoutPlus. After layout has been done, the design is (finally) ready for fabrication.

5.4 Architectural modifications vs. MOVEASAINT

Originally, the new MOVE processor generator was meant to produce architectures as similar to the ASA version as possible, for reasons of compatibility and comparability. Some modifications have been made, however. The most important changes will be discussed in this section. Some minor changes have also been made, but they won't be discussed in detail in this report. They mostly concern the removal of seemingly illogical constructs that had to be implemented in the previous processor generator because of strange limitations and bugs in the ASA software. Two new functional units have been added to the processor generator since the version described in [Lam93]. These units are compatible with the same units that have been added recently to the ASA processor generator.

5.4.1 Register file versus single register

The original MOVEASAINT architecture only features single register FUs. Though these registers are available for read and write operations at all times, the necessity for separate input and output sockets for each register makes this approach very costly in terms of used chip area. The new processor generator also supports the register file FU, which features a (parametrised) number of registers in a single FU. Though only one register in the FU can be accessed at the same time, the FU is much smaller because all its registers share the same input and output sockets. Due to the more complex register selection logic, the register file FU is also slightly slower than the single register FU. It is left to the designer to decide which approach best suits his requirements.

5.4.2 Multiply FU

In addition to the already supported integer unit, the new processor generator also features a multiply unit. The unit's wordsize is configurable in the parameter file, but the width of the result word is limited by the MOVE buswidth, if only one result output register is used. The current unit supports two, so that a multiplication of two words of size buswidth can be done. Because multipliers tend to be a lot slower than adders, the unit has a variable latency (determined by a parameter value, currently 2). Its logic is not pipelined, so only one calculation in every two cycles can be done. For applications which use multiplication operations intensively, a pipelined multiplier may have to be implemented. Alternatively, if area is not critical, a second, unpipelined multiply FU may be added.

6 Proposed modeller structure

The aim of the modeller is to characterise as efficiently as possible a number of MOVE processor configurations. Several special properties of MOVE architectures can be exploited to this end. This chapter describes a modeller structure that can make use of these properties in a way that the processor generator in itself can not. The actual implementation of the modeller is beyond the scope of this project, which was primarily to create a new MOVE processor generator. This chapter was included to provide a basis from which modeller development can proceed further.

From the modeller requirements, as described in chapter 4, it follows that the modeller system should be divided into a number of separate, but cooperating parts (fig. 6.1). These parts are the control module, the characterisation module, the cell data supply module, the database lookup module, the approximation module, the processor generator interface module and the cell property database. They will be discussed in the following sections.

The advantage of this modular approach is that the modeller can be implemented gradually, starting with a core framework, then filling in the modules one by one as they are needed. It would be possible, for instance, to have a working modeller in which the only way that cell data could be obtained, would be through synthesis and evaluation with the processor generator. The database, database lookup module and approximation module could be implemented later.

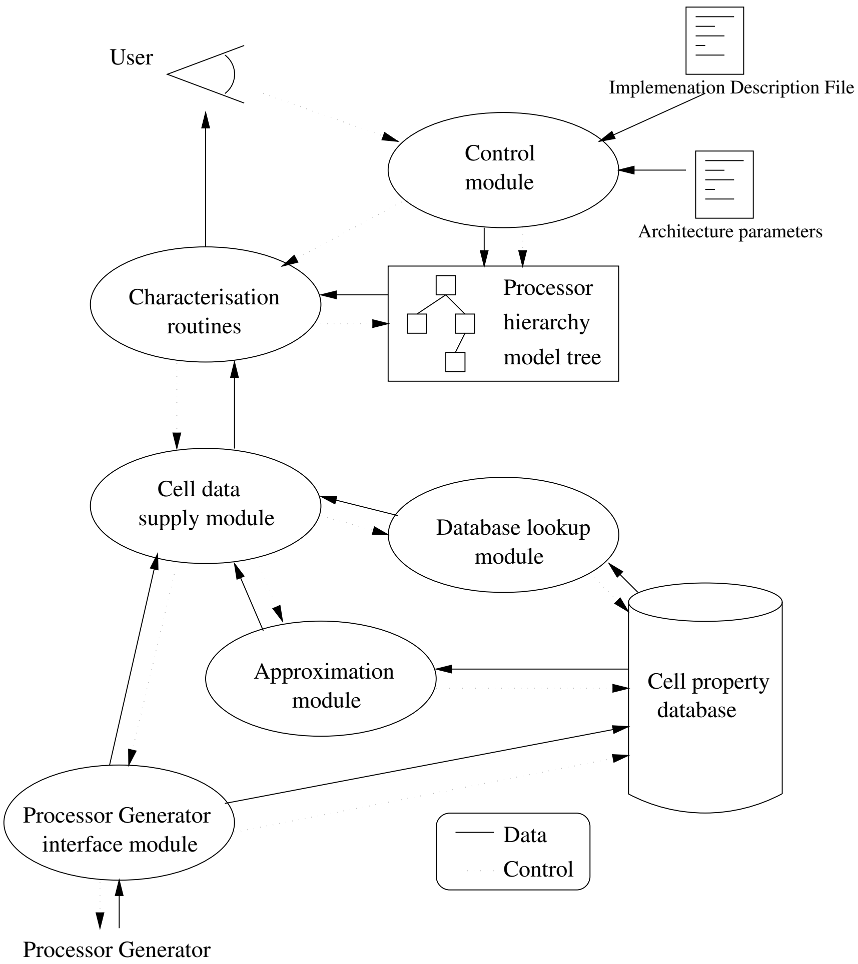


Figure 6.1: Modeller structure.

6.1 The control module

Central to the modeller is the control module. This module first sets up the instantiation tree representation of the MOVE processor hierarchy. This is the framework from which the processor is modelled, as mentioned in section 4.2. From an implementation description file (IDF, [vdL95]) and a set of architecture parameter values, the form and contents of the tree are determined. For each MOVE hierarchy design entity, a tree cell is created. Subtrees of a cell represent subsystems of the design entity. In each cell, the following information is stored:

- The cell's name.
- For each characteristic, a list of architecture parameters which influence the cell properties that are needed to calculate that characteristic. Also known as the parameter dependency list for the characteristic, it may not have to be fixed. If the

modeller can detect dependencies itself, it may want to modify the list. These are used to determine whether the relevant data has already been obtained for this configuration.

- Links to the parent cell and the subtrees, if applicable. These are used by the characterisation module to traverse the hierarchy.

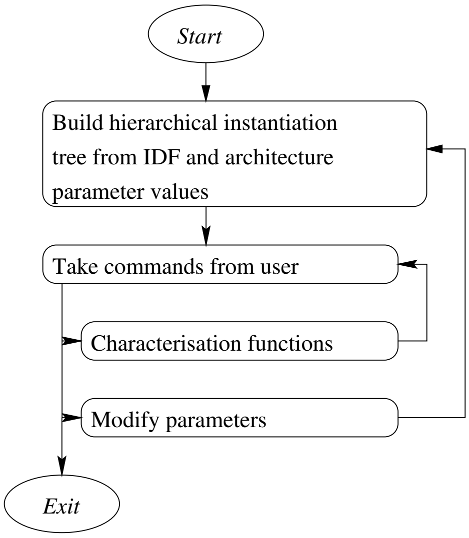


Figure 6.2: The modeller control module flowgraph

Next, the modeller awaits commands from the user. These commands fall into two categories: characterisation commands and architecture parameter modifications.

- Characterisation commands are calls to the characterisation functions. After a characterisation function has yielded a result, the modeller returns control to the user.

- Architecture parameter modifications are the means by which the user can change the MOVE processor configuration. These modifications affect the structure of the processor's internal hierarchy. It is for this reason that after such changes, a new tree must created from the implementation description and the new parameter values. Effectively, the modeller restarts.

6.2 The characterisation module

This module contains routines that each determine one of the MOVE processor configuration's characteristics. Which characteristics can be calculated depends on which routines have been implemented.

The functions have access to the modeller's tree which they can use as a reference framework to request information about the subdesigns (fig. 6.3). These requests are serviced by the data supply module (section 6.3).

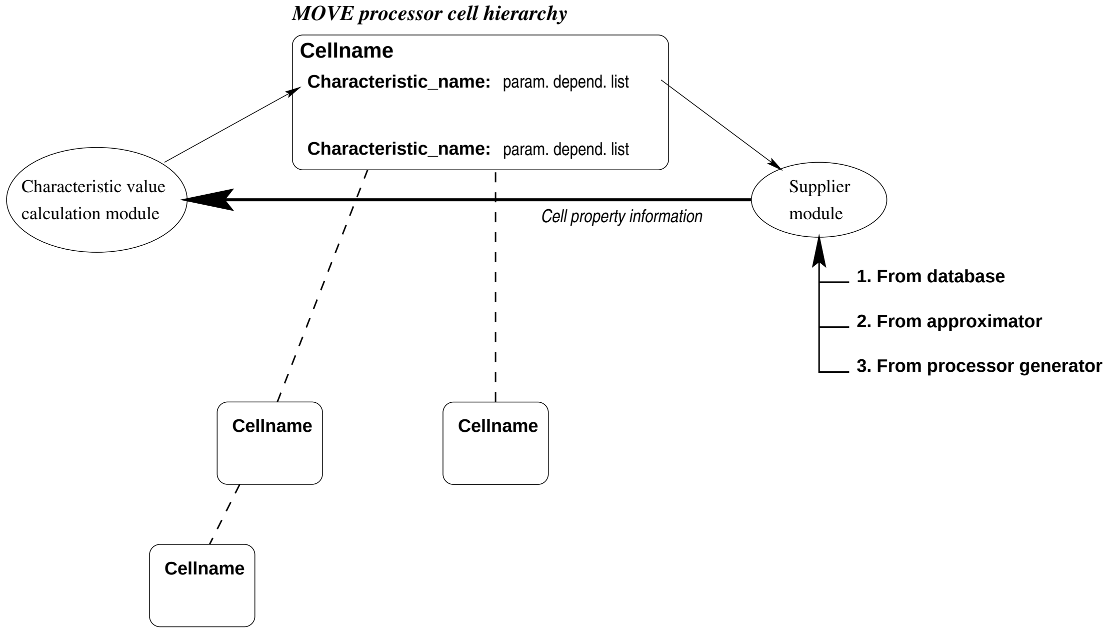


Figure 6.3: The use of the hierarchy tree to request cell data.

After the characterisation routine has gathered all the data it needs, it calculates the characteristic's value and returns it to the user.

How each characteristic is calculated depends, of course, strongly on the data that is available for each subdesign. This, in turn, depends on the data that can be obtained by the processor generator interface module. This module is different for each processor generator. It has been kept a separate part of the modeller, to keep the modeller as processor generator independent as possible. It is for this reason that the characterisation functions, which are closely related to the processor generator, as stated above, are also kept as separate as possible. An interesting side effect of this independency is that it is easy to modify the implementation of a characterisation function, provided it uses the same subdesign data. If a new function is implemented however, or if different subdesign data is used (which, effectively, is the same thing), then matters are more complicated. Each cell in the tree must contain a new entry entry for the new function, with the parameter dependency list (section 6.1). This can only be done by changing the contents of the IDF and rebuilding the tree.

Research about the actual implementation of characterisation functions has been done for the MOVE*mate* project, which is described in [vdL95].

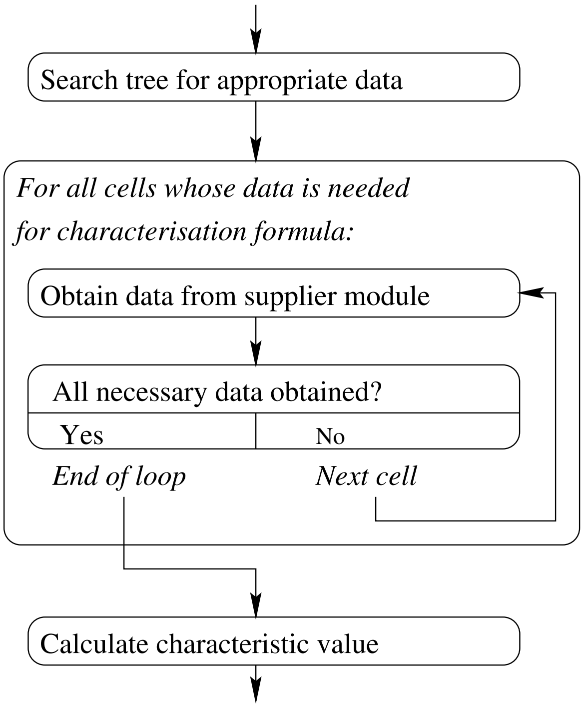


Figure 6.4: The characterisation module flowgraph.

6.3 The cell data supply module

This is the part that receives request for subdesign information from the characterisation module. These requests are in the form of the name of the characterisation function that makes the request and a cell name. The function name is used later to determine which data should be obtained for the subdesign under consideration (identified by the cellname).
The data about the subdesign can be obtained in basically three different ways, as was already described in section 4.2: by database lookup, by approximation and by synthesis with the processor generator.
First the database (6.7) is searched. In the appropriate database (there is a different one for each characterisation function) a search is done for an entry corresponding to the cellname and the values of the dependency parameter values. If no exact match is found, then the database lookup has failed.
Secondly, the approximator module (6.5) is called upon. If an approximation can be found with an acceptable accuracy, then the resulting value(s) are used.
If the approximator fails as well, then there's no other option but to call the processor generator interface module (6.6). This is by far the slowest alternative, but after it has completed, all information that can be extracted from the subdesign (not just for one characterisation function, but for all!) is available. After all, the bulk of the work is synthesising the subdesign. When this is done, it takes relatively little time to determine all

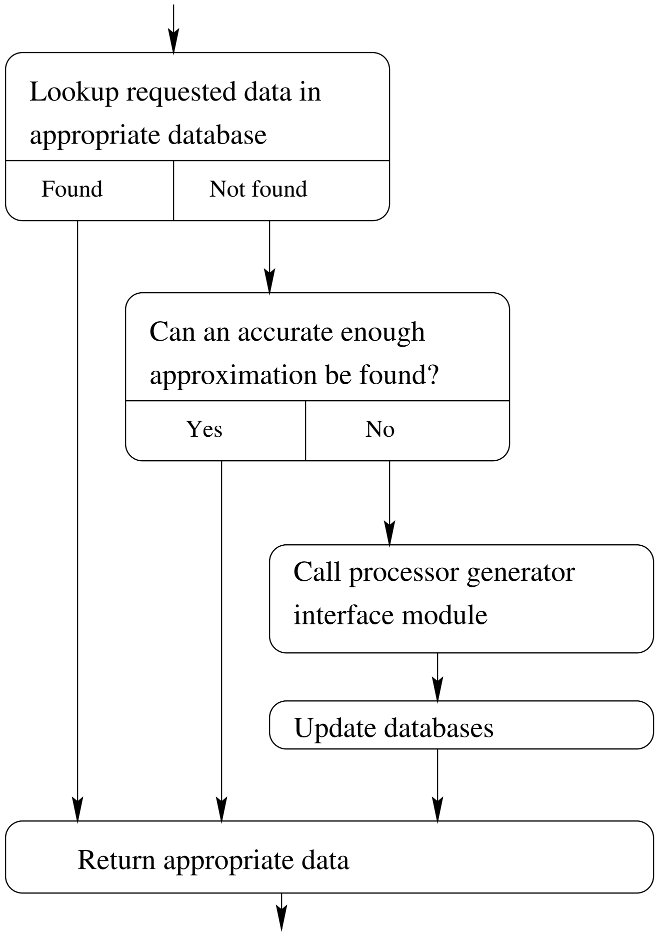


Figure 6.5: The cell data supply module flowgraph.

the subdesign's properties instead of just the ones needed at the moment. It makes sense not to waste this extra information, but to update all the appropriate databases (for the different functions) with it.

6.4 The database lookup module

This module takes requests for information from the data supply module (section 6.3. These requests are in the form of a characterisation function name, a cellname (from the hierarchy tree) and a set of parameter values. Only values of the architecture parameters that affect this particular cell's properties with regards to the current characterisation are given.

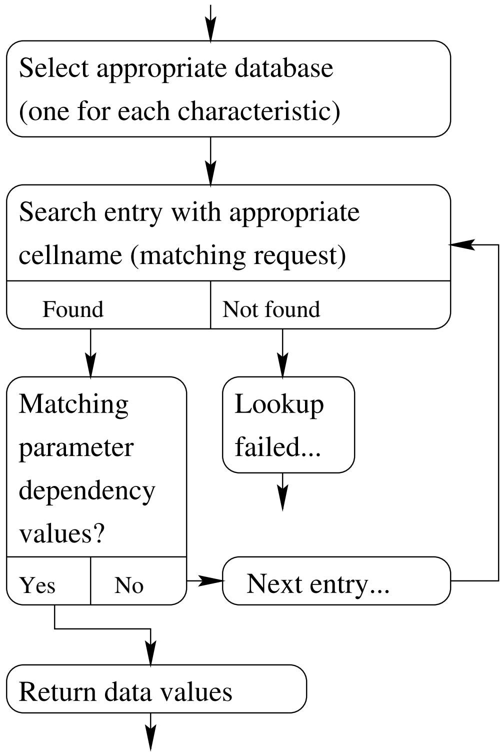


Figure 6.6: The database lookup module flowgraph.

The characterisation function's name is used to determine which database to access. The cellname and parameter values are used to search for the appropriate entry. If the entry is found, then the values of its properties, which had been stored in the database earlier, are returned to the data supply module, which in turn returns them to the characterisation module. If no entry can be found with matching cellname and dependency parameter values then the lookup has failed.

6.5 The approximator module

The approximator tries to find an approximation for the requested data values if they can't be found in the database. This is done by comparing all entries with the appropriate cellname. If a relation is detected between dependency parameter values and data entry values, then the strength of the relation should be calculated. From this it is decided whether the approximation will be accurate enough. If this is the case, then the actual approximation is calculated and the result is returned to the data supply module. If not, then the approximator has failed.

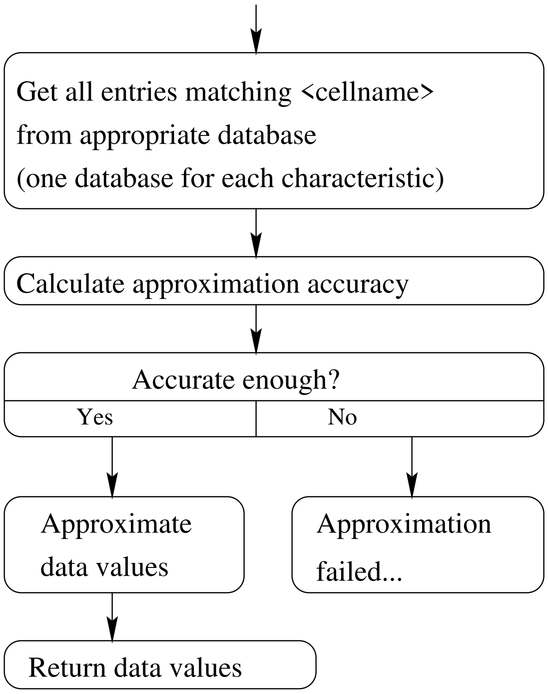


Figure 6.7: The data approximation module flowgraph.

They best way to calculate the relations and actual approximations may be different for different data entries. In some cases this can be analytical, in some cases numerical or statistical. Depending on the implementation of the approximator, several approaches could be tried subsequently. Dependencies on architecture parameters could also be detected statistically.
Note again that it is essential that data approximations are not stored in the databases. This would lead to approximations based on approximations based on etc..., rendering it impossible to give a realistic estimate of the accuracy of the approximations.

6.6 The processor generator interface module

Using the cellname and the current architecture parameter values, this module generates input and command files for the processor generator. Directed by these files, the processor generator then synthesises the subdesign (corresponding to the cellname from the hierarchy tree) and reports its properties to the interface. The interface then converts this -processor generator dependent- properties data into the format the modeller uses. In section 6.3 it was already remarked that all property data is available to create database entries for all characterisation functions.

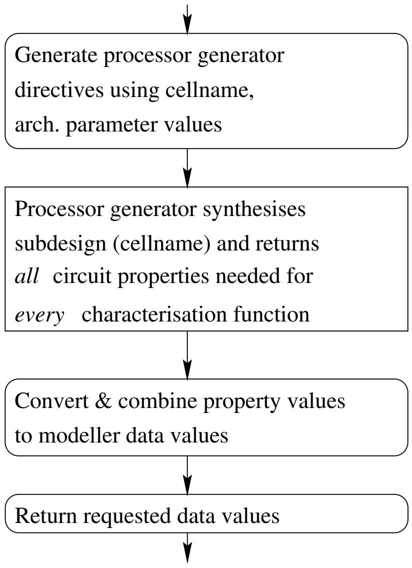


Figure 6.8: The processor generator interface flowgraph.

For reasons of database efficiency, as well as approximator performance, it would seem desirable to do as much modelling (abstraction) calculations beforehand as possible. The result would be less modeller data entries per subdesign. Less data entries means a smaller, and faster, database. Furthermore, less different data entries to find an approximation for (concurrently!) increases the chances that an accurate enough approximation is found. On the other hand, less available data to manipulate leaves less options for the implementation of the characterisation functions. This may affect the quality of the characterisation results. The effects of this trade-off are likely to be so complicated that they can only be determined experimentally.

6.7 The database

In the database, the subdesign information (obtained from the processor generator) is stored for later use, referenced by cell name. The database itself really consists of separate databases for each of the characterisation functions. This has been done for several reasons:

- To keep the number of architecture parameters each database entry depends on to a minimum. If the data for all functions would be stored in the same entry, then each entry would depend on the parameter dependency lists of all functions combined.

- To keep each characterisation function independent of the others. If the database would be shared, then adding a function would cause all the old database entries (the ones without data for the new function) to become obsolete.

Searches can only be done by cellname. It might seem more efficient to use a search by both cellname and dependency parameter values, but this is only applicable in the case of a database lookup 6.4. The approximator module, however, needs all entries for a given cellname, regardless of parameter values.

7 Experimental synthesis results

In this chapter, an evaluation of some designs made with the new processor generator can be found. They will be compared with existing MOVE architectures, the MOVE32INT [Cor93] and the MOVEASAINT [Lam93].
Due to the ongoing battle with the forces of bureaucracy, it is at this moment not possible to use the Cadence Design Framework II software. This means that no layout can be generated yet. The results presented in this chapter could therefore only be based on the gate-level description of the design as generated with the Synopsys software.
Three designs have been made with the new processor generator. The first, called MOVESYNOPSINT, is as accurately as possible a copy of the MOVEASAINT. This means ten register FUs instead of the register file FU, and no multiply FU. The second and third designs, called MOVESYNOPSINT+, have been enhanced with the register file and the multiply FU. They are 1.0 μm and 0.7 μm versions of the same design.
In section 7.1, an evaluation is presented of the chip area usage of the new designs. Section 7.2 contains their timing characteristics and the critical paths. Where possible, a comparison will be made with the existing MOVE designs.

7.1 Area usage evaluation

Since no placement and routing can be done yet, the only indication of the area of the design is the gate-level estimate. The gate-level area data for the three Synopsys-generated designs is shown in table 7.1. The calculation includes the instruction cache RAM area and the bonding pads (dimensions taken from [ES294b] and [ES294a]). Power and ground pads have not been included in the calculation. Because the ES2 wire load model does not contain area data, no approximation of the wiring area can be given.

	MOVESYNOPSINT 1μ0	**MOVESYNOPSINT+ 1μ0**	**MOVESYNOPSINT+ 0μ7**
Comb. area	5233052	5702755	3670515
Seq. area	6565949	4168802	2668033
RAM area	1397745	17277351	11057505
I/O pad area	8399998	8713058	8046741
Total area	21596743	35861966	25442781
Area in mm^2	21.6	35.9	25.4

Table 7.1: Gate-level design area in μm^2.

The only area data available for the previous MOVE32INT implementations is at layout-level. The gate-level dimensions of the new designs can not be realistically compared to this layout-level area. They can, however, be compared with eachother.
The main differences between the MOVESYNOPSINT and the MOVESYNOPSINT+ 1.0 designs are:

- The MOVESYNOPSINT has two integer FU's, whereas the MOVESYNOPSINT+ has one integer FU and one multiply FU. The multiply unit (area: 1.9 mm^2) is considerably larger than the integer unit (area: 1.0 mm^2).

- Both processors have ten registers, but the MOVESYNOPSINT has ten separate register FU's, the MOVESYNOPSINT+ only one register file FU. Ten register FU's are considerably larger than one register file FU (10×0.4 vs. 2.1 mm^2).

- The MOVESYNOPSINT's instruction cache has only 32 lines, the MOVESYNOPSINT+ has a 512 line cache. This results in a 12 times as large area for RAM (not 16, because the address decode and select logic and the precharge circuitry have almost the same area for both designs).

It is obvious that the 0.7 μm CMOS implementation of the MOVESYNOPSINT+ takes up less area than the 1.0 μm version, about 30%. All gates within the die are about 36% smaller, the difference is caused by the I/O pads (only 8% smaller).

7.2 Timing evaluation

Because there's no layout available yet for the new processors, all timing analysis will also have to be done at gate-level. Fortunately, gate-level timing data is also available for the MOVEASAINT [Lam93], so at least some comparison can be done. The timing characteristics for the new and previous MOVE designs can be found in table 7.2.

		Layout level		Gate level	
Processor	**Process**	**Delay**	**1/delay**	**Delay**	**1/delay**
MOVE32INT	SoG (1.6 μm)	12.5 ns	80 MHz		
MOVEASAINT	ES2/CMS1M5			27.7 ns	36.1 MHz
MOVEASAINT	ES2/CMS1M0	23.3 ns	43 MHz	19.5 ns	51.2 MHz
MOVESYNOPSINT	ECPD10			14.9 ns	67.1 MHz
MOVESYNOPSINT+	ECPD10			17.9 ns	55.8 MHz
MOVESYNOPSINT+	ECPD07			15.7 ns	63.7 MHz

Table 7.2: Processor timing characteristics.

When comparing the MOVEASAINT with the MOVESYNOPSINT, which closely resembles it in both process (1 μm) and architecture (MOVE32INT compatible), it can be seen that the new design is considerably faster (over 30%). This is most likely caused by the more extensive optimsation possibilities the Synopsys Design Compiler [Syn94b] offers. For instance, the Design Compiler can do boundary optimisation, which is optimisation of paths accross subdesign boundaries. ASA's optimisations can only be done within each cell, independently of the other cells.
The critical paths (gate level) for the new designs are given in tables 7.3, 7.4 and 7.5. Note that the multiplier combinational logic is two-cycle!

Exception path	**Cummulative delay**
compare FU result	2.02 ns
guard unit	6.31 ns
squashlines	6.31 ns
integer FU exc. logic	9.13 ns
netcontrol	10.84 ns
ifetch FU PC	14.86 ns

Table 7.3: MOVESYNOPSINT gate level critical path.

In the MOVEASAINT design, the critical path ran through the integer FU. The MOVESYNOPSINT obviously has a faster adder implementation, since this path is no longer critical. Its exception path is the critical one. This path is also quite long (18.0 ns) in the MOVEASAINT version and would have been critical there, too, if it hadn't been for the integer FU path.
The critical path for the MOVESYNOPSINT+ (1.0 μm) runs through one of the new units: the multiplu FU. The complexity of the combinational logic made it necessary to make this a two-cycle path. The path length of 35.82 ns then results in a 17.9 ns cycle. It should be noted that increasing the multiplier logic's path to three cycles will not improve the timing of the design. The next critical path is only marginally shorter, just

Multiply FU Path	Cummulative delay
multiply FU operand reg.	3.69 ns
multiplier combinational logic	34.85 ns
result register	35.82 ns

Table 7.4: MOVESYNOPSINT+ 1.0 μm gate level critical path.

0.18 ns. This path is the suqash path; it runs through the compare unit, the valid unit, the squashlines and finally the register file. It is the complicated structure of the register file (a lot of selection logic) that is responsible for most of the delay. Because this is a global path (transcending FU boundaries), the effects of placement and routing will probably be more serious on this path than on the current (gate-level) critical path, which is restricted to one FU. It is therefore likely that the squash path to the register file FU will become the critical path at layout-level.

Global lock path	Cummulative delay
integer FU local lock req.	7.05 ns
netcontrol global lock	10.57 ns
multiply FU output register 2	17.52 ns

Table 7.5: MOVESYNOPSINT+ 0.7 μm gate level critical path.

The critical path of the MOVESYNOPSINT+ (0.7 μm) is also a global one. The effect of placement and routing on these paths is extremely difficult to predict beforehand. The delay of these paths depends largely on the delay of the wires that connect the different FU's accross the chip, more so than in the case of local paths, which stay inside a single FU cell. Because nothing is known beforehand about the placement of the FU's on the actual chip, it's difficult to calculate the wire delays. Synopsys solves this problem by introducing a wire load model [Syn94b]:

> *"In the absence of physical design information, Design Compiler uses statistically generated wire load models to estimate wire lengths in a design. The wire load models, provided in the technology library, define a fanout to length relationship."*

It is difficult to predict how representative this approximation will be of the layout-level path delays, but it's likely to be more accurate for local paths than global paths. The only way to verify this, however, will be generating and evaluating actual layout for the designs.

8 Conclusions

A new processor generator using the Synopsys silicon compiler has been implemented. From a set of template VHDL description files and a user-defined architecture parameter file, a complete, synthesisable processor description is generated with the macro preprocessing tool m4. The initial requirements (MOVE architecture, MOVEASAINT compatible, automatic scalability, easy architectural modification and adding of functionality) have been met, as well as the requirement which surfaced later: the option to generate partial designs (for the modeler). The gate level designs can be exported from Synopsys Design Compiler to the Cadence Design Framework II, which is used to transform it to layout.

In order to compare the performance of the new processor generator with the previous (ASA) version, a design has been generated which closely resembles the MOVEASAINT implementation in both architecture and process technology. Both designs use the ES2 1.0 μm CMOS process and are MOVE32INT compatible. Gate-level analysis reveals the design generated by the new processor generator to be 30% faster. The estimated clock frequency is 67 MHz, as opposed to 51 MHz for the MOVEASAINT implementation. The most plausible explanation for this increase in speed is the fact that the Synopsys Design Compiler can do critical path optimisation across subdesign boundaries.

In addition, two MOVE designs have been generated that incorporate both new features: a multiply FU and a register file FU. The designs are implemented in the ES2 0.7 μm CMOS and 1.0 μm CMOS processes and run at an estimated speed of 64 and 56 MHz, respectively. The reason for this relative loss of speed (when compared to the

MOVE32INT compatible version) is the increased complexity of the circuitry needed for the added functionality.
A modeler system was specified, which was designed to evaluate large sets of MOVE processor configurations in an efficient manner. It exploits the potential for (partial) evaluation data reusability among different, but similar, MOVE configurations. Data approximation is supported to increase the modeller's efficiency. The characterisation module has been designed for adaptability: characterisation routines can be modified or added without having to change the entire modeler structure. Because of its modular structure, the modeler can implemented gradually, starting with a core framework to which extra functionality can be added later.

Bibliography

[Arm88] J. Armstrong. *Chip Level Modelling in VHDL*. Prentice Hall, 1988.

[BFMR92] Jean-Michel Berge, Alain Fonkoua, Serge Maginot, and Jacques Rouillard. *VHDL Designers Reference*. Kluwer Academic Publishers, 1992.

[Cad94] Cadence Design Systems Inc. *Cadence Design Framework II Reference Manual*, 1994.

[CM91a] H. Corporaal and J.M. Mulder. MOVE: A flexible and Scalable Architecture for High-Performance Processor Design. Patent Application, April 1991.

[CM91b] Henk Corporaal and Hans (J.M.) Mulder. MOVE: A framework for high-performance processor design. In *Supercomputing-91*, pages 692–701, Albuquerque, November 1991.

[CM92] Hendrik Corporaal and Johannes Martinus Mulder. Processor employing a parallel transport triggered architecture. Patent Application, April 1992.

[Cor93] Henk Corporaal. MOVE32INT, Architecture and Programmer's Reference Manual. Technical Report 1-68340-44-(1993)01, Delft University of Technology, Electr. Eng. Department, The Netherlands, 1993.

[Cor94a] Henk Corporaal. Design of Transport Triggered Architectures. In *Fourth Great Lakes Symposium on VLSI*, pages 130–135, Univ. of Notre Dame, Indiana, March 1994.

[Cor94b] Henk Corporaal. Transport Triggered Architectures. In *GI/ITG-workshops on Architecturen für hochintegrierte Schaltungen*, pages 33–35, Schloss Dagstuhl, July 1994.

[CvdA93] Henk Corporaal and Paul van der Arend. MOVE32INT, a Sea of Gates realization of a high performance Transport Triggered Architecture. *Microprocessing and Microprogramming*, 38:53–60, September 1993.

[ES293a] European Silicon Structures. *ES2 Cadence Design Kit User Guide*, 1993.

[ES293b] European Silicon Structures. *ES2 Synopsys Design Kit User Guide*, 1993.

[ES294a] European Silicon Structures. *ES2 ECPD07 Library Databook*, 1994.

[ES294b] European Silicon Structures. *ES2 ECPD10 Library Databook*, 1994.

[ES294c] European Silicon Structures. *ES2 Engineering Application Notes*, 1994.

[ES294d] European Silicon Structures. *ES2 FAST User Guide*, 1994.

[Goo89] A.J. van de Goor. *Computer Architecture and Design*. Addison Wesley, 1989.

[GS92] Patrick Groeneveld and Paul Stravers. The seafood menu for layout: Seadali, fish, trout, madonna and more. Technical report, Electronal Engineering faculty, Delft University of Technology, November 1992.

[HC94] Jan Hoogerbrugge and Henk Corporaal. Transport-triggering vs. operation-triggering. In *Lecture Notes in Computer Science 786, Compiler Construction*, pages 435–449. Springer-Verlag, 1994.

[Hof94] Maarten Hofman. Design and modeling of MOVE processors using VHDL and COMPASS. Master's thesis, Delft University of Technology, April 1994.

[HP90] John L. Hennessy and David A. Patterson. *Computer Architecture, a quantitative approach*. Morgan Kaufmann publishers, 1990.

[HS91] Randolf E. Harr and Alec G. Stanculescu. *Applications of VHDL to Circuit Design*. Kluwer Academic Publishers, 1991.

[IEE87] IEEE Standards, The IEEE Computer Society Press. *1076-1987 Standard Language Reference Manual*, 1987.

[Lam93] Reinoud Lamberts. A move processor generator. Technical report, Delft Univ. of Technology, November 1993.

[M$^+$89] J.M. Mulder et al. A framework for application-specific architecture design. In *Proceedings of the 16th International Symphosium on Computer Architecture*, May 1989.

[M4,92] Hewlett-Packard Company. *HP-UX Release 9.0: The M4 Macro Preprocessor*, 1992.

[Mic94] Giovanni De Micheli. *Synthesis and optimization of digital circuits*. McGRAW-HILL, 1994.

[Nav92] Zainalabedin Navabi. *VHDL:Analysis and Modeling of Digital Systems*. McGraw Hill, 1992.

[Per90] Doug Perry. *VHDL*. McGraw Hill, 1990.

[Sag93] Sagantec Eindhoven. *ASA version 5.4 manuals*, 1993.

[Sch91] Sietze Schukking. MOVE in real world surroundings: three proposals. Technical Report 1-68340-(1992)2, Delft University of Technology, December 1991.

[Str91] Paul Stravers. Fishbone. Technical Report ?, Electronal Engineering faculty, Delft University of Technology, 1991.

[Syn94a] Synopsys Inc. *Synopsys Design Analyzer Reference Manual*, 1994.

[Syn94b] Synopsys Inc. *Synopsys Design Compiler Reference Manual*, 1994.

[Syn94c] Synopsys Inc. *Synopsys DesignWare Documentation*, 1994.

[vdL95] Marcel van der Laan. MOVE*mate*, A *M*ultifunctional *A*rea and *T*iming *E* stimator for the MOVE framework . Master's thesis, Delft University of Technology, May 1995.

Functional Units

A

This appendix contains details on the new functional units supported by the MOVE Processor Generator: the multiply and register file FU's. Information about the other units can be found in [Lam93], chapter 4. Every one of the following sections describes one FU type. The sections each cover the following topics, when applicable:

- **General:** A short description of the function of the FU.
- **Parameters:** The architecture parameters that influence the structure of the FU.
- **Structure:** A graphical overview of the FU structure.
- **Operations:** The operations the FU supports.

  The **Operation** part shows the move operation in the standard format $g(r)\ :\ A_{src} \rightarrow A_{dst}$, where $g(r)$ is a guard specifier that depends on r, $A_{src} \rightarrow A_{dst}$ is the move from the source address that belongs to *src* to the destination address that belongs to *dst*. Addresses of sockets are specified with the socket names used in pipeline diagrams (e.g. A_T). Operations indicated with *-implicit-* are not coupled directly to move operations, but are performed at times dictated by, for example, (hybrid) pipeline control.

  The **Semantics** part describes the effect of the operations. The notation used is as in $v\ :\ r \leftarrow f_{int}(op, t, o)$, where $\leftarrow$ indicates an assignment, and most of the identifiers used are names of registers, control state or sockets which can be found in the pipeline diagram. In the example, an (integer unit) function of the opcode, the trigger register and the operand register is assigned to the result register. The assignment is 'guarded' by the valid bits v. Opcodes are described symbolically in square brackets, e.g. $[t + o]$ for the 'add trigger register and operand register' opcode. Often used function subscripts include unit-specific ones (such as *int* for integer unit) and *hybrid* to indicate the hybrid pipeline control function. Hybrid pipelining details can be found in [Lam93], chapter 5.

- **Control support:** An overview of supported locks and exceptions.
- **Notes:** Additional information.

A.1 Multiply units

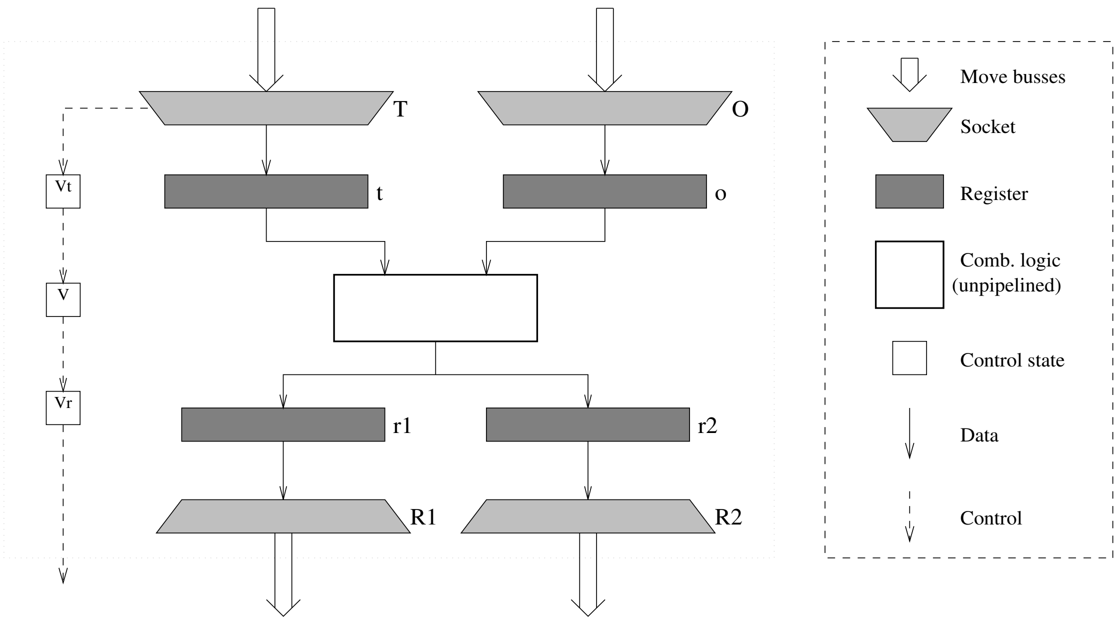
General

Multiply units perform multiplication on parts of the trigger and operand register values. The size of the parts is defined by the architecture parameter MULFUWORDSIZE. Because of the complexity of the multiplier logic, this unit usually has a MULFUDELAY of 2.

Multiply unit parameters

BUSWIDTH	Register size (external)
MULFUS	Number of instantiations
MULFUDELAY	Delay cycles for multiplier logic
MULFUWORDSIZE	Multiplier word size (internal)

Multiply unit structure



Multiply unit operations

Operation	Semantics
-implicit-	$v \leftarrow f_{hybrid}(v, T, R), r \leftarrow f_{mul}(op, t, o)$
$A_{src} \rightarrow A_T$	$t \leftarrow src, op \leftarrow [t * o]$
$A_{src} \rightarrow A_O$	$o \leftarrow src$
$A_{R1} \rightarrow A_{dst}$	$dst \leftarrow r1$
$A_{R2} \rightarrow A_{dst}$	$dst \leftarrow r2$

Multiply unit control support

Socket	Lock support	Exception support
T	write	pipeline full
O	write	pipeline full
R1	read	pipeline empty
R2	read	pipeline empty

A.2 Register file unit

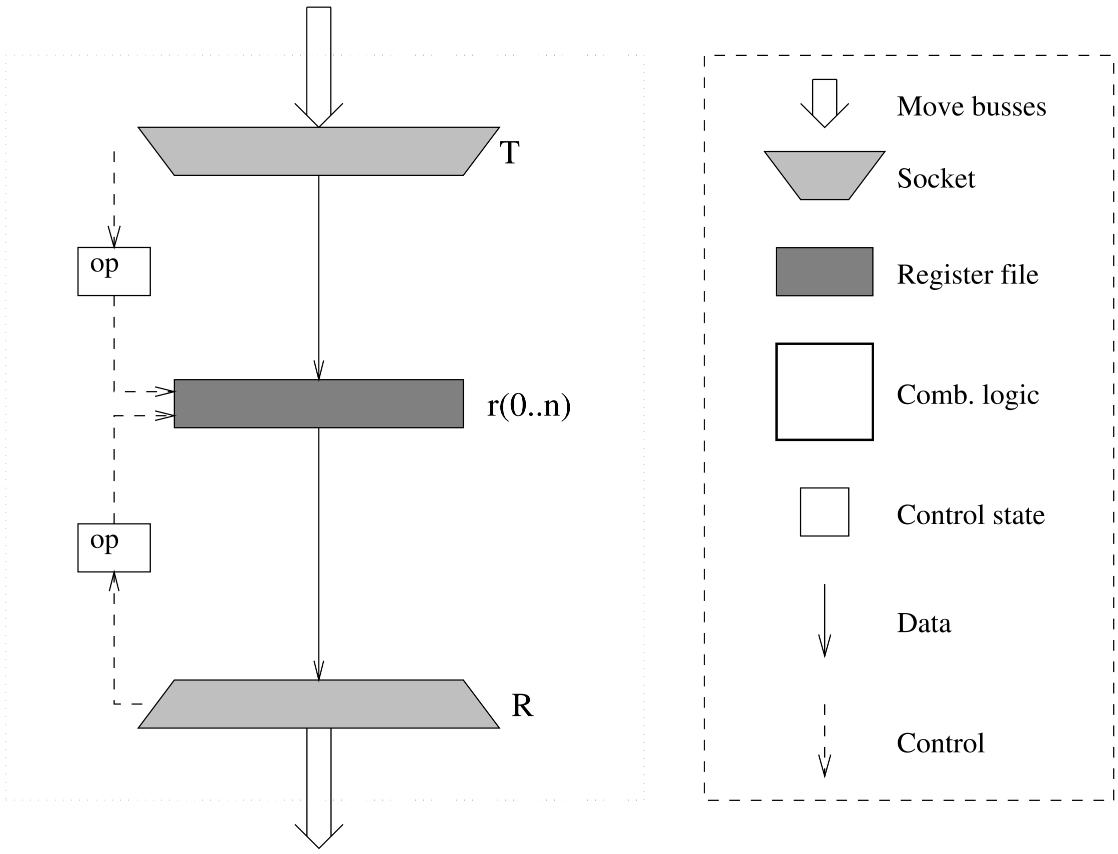
General

The register file unit contains one or more general purpose registers. One read and one write are permitted concurrently each cycle. If the RFLBYPASS parameter is set, an early read selection is done and a bypass is used. This avoids increase of move stage delay due to register selection, but requires more area.

Register File unit parameters

BUSWIDTH	Register size
RFLFUS	Number of instantiations
RFLBYPASS	Define whether bypass should be used
PRFLFUx_REGS	Number of registers in FU x
PRFLFUx_IDS	Number of socket IDs

Register File structure



Register File operations

Operation	Semantics
$A_{src} \rightarrow A_{T+n}$	$r_n \leftarrow src$
$A_{R+n} \rightarrow A_{dst}$	$dst \leftarrow r_n$

Note: The register file contains n registers.

Register File unit control support

Socket	Lock support	Exception support
T	–	–
R	–	–

Architecture parameters B

In this appendix, the `params_user` file has been included. It contains the architecture parameter declarations that are specified by the user before generating a design with the MOVE processor generator.
The file has been kept very similar to the `parmdecl.sih` file of the previous processor generator [Lam93].

```
changequote
changequote({,})

#------------------------------------------------------------------------------
# User parameter declarations file
# Enhanced 32 bit version
#
# The parameters below define the processor characteristics. Unless you want
# to add functionality, only this file needs to be changed to customize
# the processor.
#------------------------------------------------------------------------------
#
#
#   Connection defines
#
#   Most units have sockets connecting to move busses. For each socket it
#   can be specified to which of the busses it will connect. This is done
#   by defines such as the following:
#
#     define({CINTFU1_TBUS0}, 0)
#     define({CINTFU1_TBUS1}, 0)
#     define({CINTFU1_TBUS2}, 1)
#     define({CINTFU1_TBUS3}, 1)
```

```
#
#   In this example, the trigger socket connections for the second
#   integer FU are defined, for a processor with PARMOVES == 4. The
#   socket will connect to busses 2 and 3, but not to 0 and 1.
#
#   For other FUs and sockets, the parameter naming is similar, see
#   further below in this file. The fields in the identifiers are as
#   follows:
#
#       CINTFU1_TBUS2
#       |\   /|||\ /|
#       | \ / ||| V |
#       |  V  ||| | Bus number
#       |  |  ||| |
#       |  |  ||| Always ’BUS’
#       |  |  |||
#       |  |  ||Socket selector: T->trigger, O->operand, R->result
#       |  |  ||selectors can have subscripts (e.g. ’R2’)
#       |  |  ||
#       |  |  |Always ’_’
#       |  |  |
#       |  |  FU number, starting at 0
#       |  |
#       |  FU type
#       |
#       Always ’C’ (for Connect)


#-----------------------------------------------------------------------------
#  Primary Processor Characteristics
#-----------------------------------------------------------------------------

#   BUSWIDTH: defines the data (move-) bus width, and implies maximum address
#   pointer size. It does not define the number of external address bits.
#   Must be >= 3, because of the shift unit implementation.


define({BUSWIDTH}, 32)

#   PARMOVES: number of move busses. Equals the number of move operations
#   in an instruction in this implementation.

define({PARMOVES}, 4)

#------------------------------------------------------------------------------
# Ifetch FU
#------------------------------------------------------------------------------

#    IFESHIFTADDR: if defined, instruction address registers in the
#    ifetch unit are connected to the buses such that they map directly
#    to data addresses; the lower bits are ignored on writes and return
#    zero on read. If undefined, the lowest instruction address bit
#    connects to bit 0 of the bus.
#
#    Instructions are assumed to be aligned to data word boundaries in
#    external memory, with no more than 1 instruction stored per data
#    word.
#
#    Note that using shifted address registers reduces the number of
#    immediate bits than can be used to specify an address (offset)
#    directly. This may also influence the exception handler.


undefine({IFESHIFTADDR})
```

```
#    IFEADDRSIZE: instruction address size. Instruction space is
#    limited by BUSWIDTH for address register size if IFESHIFTADDR
#    is undefined. If IFESHIFTADDR is defined, maximum instruction
#    address size is reduced by the number of bits shifted.
#
#    You can leave IFEADDRSIZE undefined if IFEAUTOADDRSIZE is defined.

#define({IFEADDRSIZE}, 16)

#    IFEAUTOADDRSIZE: if defined, automatically choose IFEADDRSIZE to
#    get instruction and data spaces of the same size; you can set
#    IFEADDRSIZE yourself if IFEAUTOADDRSIZE is undefined. Automatic
#    IFEADDRSIZE determination assumes no more than 1 instruction is
#    stored per data word (BUSWIDTH); an instruction may take several
#    data words in memory though. Alignment of instructions to word
#    boundaries is taken into account.
#
#    Works independently from IFESHIFTADDR.

define({IFEAUTOADDRSIZE})

#    IFEOFFSETSIZE: number of 'pco' offset bits. Note that this parameter
#    determines the number of offset bits actually used in an instruction
#    address, so if the lower address bits are unused (see IFESHIFTADDR),
#    the offset bits are shifted to the left. Therefore you may want to
#    set this value to less than the number of immediate bits.
#
#    Requirement: IFEOFFSETSIZE < IFEADDRSIZE.

define({IFEOFFSETSIZE}, 6)

#    IFEFUDELAY: number of cycles for instruction fetch from external
#    memory (strobe and latch cycles inclusive). Must be >= 2.

define({IFEFUDELAY}, 2)

define({IFERSTVEC}, 000001000000)
#   reset vector
define({IFEEXCVEC}, 000010000000)
#   exception vector
define({IFEINTVEC}, 000011000000)
#   interrupt vector

#  IFECACHESIZE: number of cache lines, one instruction per line. Must
#   be a power of two, 4 <= IFECACHESIZE <= 2048. The maximum allowed
#   cache size is more than 64 kbyte (including tag). The number of
#   cache lines may be at most half the number of instruction addresses.

define({IFECACHESIZE}, 32)

#  Connection defines

define({CIFEFU0_TBUS0}, 1)
define({CIFEFU0_TBUS1}, 0)
define({CIFEFU0_TBUS2}, 1)
define({CIFEFU0_TBUS3}, 0)

define({CIFEFU0_R1BUS0}, 1)
define({CIFEFU0_R1BUS1}, 1)
define({CIFEFU0_R1BUS2}, 0)
define({CIFEFU0_R1BUS3}, 0)

define({CIFEFU0_R2BUS0}, 1)
```

```
define({CIFEFU0_R2BUS1}, 1)
define({CIFEFU0_R2BUS2}, 0)
define({CIFEFU0_R2BUS3}, 0)


#----------------------------------------------------------------------------
#  Guard Unit
#-----------------------------------------------------------------------------

#    SINGLEGUARD: only one guard signal is used if defined, capable of
#    squashing the entire instruction. If undefined, each move bus is
#    guarded individually.

undefine({SINGLEGUARD})

#    GUARDSPECSIZE: defines the number of bits per guard specifier in
#    an instruction. Must be 3 for now.

define({GUARDSPECSIZE}, 3)


#----------------------------------------------------------------------------
#  Immediate Unit
#----------------------------------------------------------------------------

#   IMMFUS: number of immediate FUs. Using more than one is useless,
#   but possible.

define({IMMFUS}, 1)

#   IMMEDIATES: number of immediate values. Defines the range of
#   immediates as 0 .. IMMEDIATES - 1 (assuming IMMSIGNEXTEND
#   undefined). IMMEDIATES must be a power of two >= largest PRFLFUx_IDS.

define({IMMEDIATES}, 64)

#   Connection defines
#   Unlike most other units, here each bus connection defines connecting
#   a different result register to the bus (immediates are independent
#   for each bus).

define({CIMMFU0_RBUS0}, 1)
define({CIMMFU0_RBUS1}, 1)
define({CIMMFU0_RBUS2}, 1)
define({CIMMFU0_RBUS3}, 1)

#----------------------------------------------------------------------------
#  Load-Store Units
#----------------------------------------------------------------------------

define({LDSFUS}, 1)     /* number of load-store FUs */

#   LDSFUDELAY: number of cycles for data load or store operation
#   (strobe and latch cycles inclusive). Must be >= 1; address
#   strobe and data latch may overlap.

define({LDSFUDELAY}, 2)

#   LDSADDRSIZE: data address size. Data space is limited by BUSWIDTH
#   which determines maximum address register size.

define({LDSADDRSIZE}, 15)

#   LDSOLOCKFULL: full operand socket (store data) lock implementation
```

```
#   if defined: lock only if store in progress. If undefined, lock
#   always when memory operation in progress. In the latter case the
#   lock path may be shorter, but a write to the O register while a
#   load is in progress will lock the processor unnecessarily.

define({LDSOLOCKFULL})

#  Connection defines

define({CLDSFU0_TBUS0}, 1)
define({CLDSFU0_TBUS1}, 0)
define({CLDSFU0_TBUS2}, 0)
define({CLDSFU0_TBUS3}, 1)

define({CLDSFU0_OBUS0}, 0)
define({CLDSFU0_OBUS1}, 1)
define({CLDSFU0_OBUS2}, 1)
define({CLDSFU0_OBUS3}, 0)

define({CLDSFU0_RBUS0}, 1)
define({CLDSFU0_RBUS1}, 0)
define({CLDSFU0_RBUS2}, 0)
define({CLDSFU0_RBUS3}, 1)


#----------------------------------------------------------------------------
#  Compare Units
#----------------------------------------------------------------------------

define({CMPFUS}, 2)

#  Connection defines

define({CCMPFU0_TBUS0}, 1)
define({CCMPFU0_TBUS1}, 1)
define({CCMPFU0_TBUS2}, 0)
define({CCMPFU0_TBUS3}, 0)

define({CCMPFU0_OBUS0}, 0)
define({CCMPFU0_OBUS1}, 1)
define({CCMPFU0_OBUS2}, 1)
define({CCMPFU0_OBUS3}, 0)

define({CCMPFU0_RBUS0}, 1)  # not in MOVE32INT spec
define({CCMPFU0_RBUS1}, 0)
define({CCMPFU0_RBUS2}, 0)
define({CCMPFU0_RBUS3}, 1)

define({CCMPFU1_TBUS0}, 0)
define({CCMPFU1_TBUS1}, 0)
define({CCMPFU1_TBUS2}, 1)
define({CCMPFU1_TBUS3}, 1)

define({CCMPFU1_OBUS0}, 1)
define({CCMPFU1_OBUS1}, 0)
define({CCMPFU1_OBUS2}, 0)
define({CCMPFU1_OBUS3}, 1)

define({CCMPFU1_RBUS0}, 0)  # not in MOVE32INT spec
define({CCMPFU1_RBUS1}, 1)
define({CCMPFU1_RBUS2}, 1)
define({CCMPFU1_RBUS3}, 0)
```

```
#---------------------------------------------------------------------------
#  Multiply Units
#---------------------------------------------------------------------------

define({MULFUS}, 1) # number of multiply FUs

#  MULFUDELAY: determines the number of cycles the stage control
#  allows for execution of the operation.

define({MULFUDELAY}, 2)

#  MULFUWORDSIZE: size of operands and high/low results. Results
#  smaller than BUSWIDTH are sign-extended.

define({MULFUWORDSIZE}, 16)

#  Connection defines

define({CMULFU0_TBUS0}, 0)
define({CMULFU0_TBUS1}, 0)
define({CMULFU0_TBUS2}, 1)
define({CMULFU0_TBUS3}, 1)

define({CMULFU0_OBUS0}, 0)
define({CMULFU0_OBUS1}, 1)
define({CMULFU0_OBUS2}, 1)
define({CMULFU0_OBUS3}, 0)

define({CMULFU0_R1BUS0}, 0) # result low word
define({CMULFU0_R1BUS1}, 1)
define({CMULFU0_R1BUS2}, 0)
define({CMULFU0_R1BUS3}, 1)

define({CMULFU0_R2BUS0}, 1) # result high word
define({CMULFU0_R2BUS1}, 0)
define({CMULFU0_R2BUS2}, 1)
define({CMULFU0_R2BUS3}, 0)


#---------------------------------------------------------------------------
#  Integer Units
#---------------------------------------------------------------------------

define({INTFUS}, 1) # number of integer FUs

#  INTFUDELAY: determines the number of cycles the stage control
#  allows for execution of the operation.

define({INTFUDELAY}, 1) # MOVE32INT has a 2 cycle integer unit!

# Connection defines

define({CINTFU0_TBUS0}, 1)
define({CINTFU0_TBUS1}, 1)
define({CINTFU0_TBUS2}, 0)
define({CINTFU0_TBUS3}, 0)

define({CINTFU0_OBUS0}, 1)
define({CINTFU0_OBUS1}, 0)
define({CINTFU0_OBUS2}, 0)
define({CINTFU0_OBUS3}, 1)

define({CINTFU0_RBUS0}, 1)
define({CINTFU0_RBUS1}, 0)
```

```
define({CINTFU0_RBUS2}, 1)
define({CINTFU0_RBUS3}, 0)

define({CINTFU1_TBUS0}, 0)
define({CINTFU1_TBUS1}, 0)
define({CINTFU1_TBUS2}, 1)
define({CINTFU1_TBUS3}, 1)

define({CINTFU1_OBUS0}, 0)
define({CINTFU1_OBUS1}, 1)
define({CINTFU1_OBUS2}, 1)
define({CINTFU1_OBUS3}, 0)

define({CINTFU1_RBUS0}, 0)
define({CINTFU1_RBUS1}, 1)
define({CINTFU1_RBUS2}, 0)
define({CINTFU1_RBUS3}, 1)


#---------------------------------------------------------------------------
#  Logic Units
#---------------------------------------------------------------------------

define({LOGFUS}, 1) # number of logic units

#  Connection defines

define({CLOGFU0_TBUS0}, 0)
define({CLOGFU0_TBUS1}, 1)
define({CLOGFU0_TBUS2}, 0)
define({CLOGFU0_TBUS3}, 1)

define({CLOGFU0_OBUS0}, 1)
define({CLOGFU0_OBUS1}, 0)
define({CLOGFU0_OBUS2}, 1)
define({CLOGFU0_OBUS3}, 0)

define({CLOGFU0_RBUS0}, 0)
define({CLOGFU0_RBUS1}, 0)
define({CLOGFU0_RBUS2}, 1)
define({CLOGFU0_RBUS3}, 1)


#---------------------------------------------------------------------------
#  Shift Units
#---------------------------------------------------------------------------

define({SHIFUS}, 1) # number of shift units

#  Connection defines

define({CSHIFU0_TBUS0}, 0)
define({CSHIFU0_TBUS1}, 1)
define({CSHIFU0_TBUS2}, 0)
define({CSHIFU0_TBUS3}, 1)

define({CSHIFU0_R1BUS0}, 1)
define({CSHIFU0_R1BUS1}, 0)
define({CSHIFU0_R1BUS2}, 1)
define({CSHIFU0_R1BUS3}, 0)

define({CSHIFU0_R2BUS0}, 0)
define({CSHIFU0_R2BUS1}, 1)
define({CSHIFU0_R2BUS2}, 0)
```

```
define({CSHIFU0_R2BUS3}, 1)


#---------------------------------------------------------------------------
#  Valid Unit
#---------------------------------------------------------------------------

#  Connection defines
#   Each valid bit can appear on only one bus. Only as many connects
#   need be defined as the number of valid bits vs. BUSWIDTH requires.
#
#   The valid bits are assigned from the highest bus number and the
#   most significant bit downwards.

define({CVALFU_RBUS0}, 0)
define({CVALFU_RBUS1}, 0)
define({CVALFU_RBUS2}, 0)
define({CVALFU_RBUS3}, 1)


#---------------------------------------------------------------------------
#  Register File Units
#---------------------------------------------------------------------------

define({RFLFUS}, 1) # number of register file units

#   RFLBYPASS: do early register read selection and use a bypass. Defining
#   this avoids increase of move stage delay due to register selection,
#   but requires more area.

define({RFLBYPASS})

#  Connection and register count defines

define({PRFLFU0_REGS}, 10) # number of registers in FU 0
define({PRFLFU0_IDS},  16) # number of socket IDs must be a
# power of 2 and >= 8 and <= IMMEDIATES

define({CRFLFU0_TBUS0}, 1)
define({CRFLFU0_TBUS1}, 1)
define({CRFLFU0_TBUS2}, 1)
define({CRFLFU0_TBUS3}, 1)

define({CRFLFU0_RBUS0}, 1)
define({CRFLFU0_RBUS1}, 1)
define({CRFLFU0_RBUS2}, 1)
define({CRFLFU0_RBUS3}, 1)


#---------------------------------------------------------------------------
#  Register Units
#---------------------------------------------------------------------------

define({REGFUS}, 0) # number of (obsolete) register units

#  Connection defines

define({CREGFU0_TBUS0}, 0)
define({CREGFU0_TBUS1}, 0)
define({CREGFU0_TBUS2}, 1)
define({CREGFU0_TBUS3}, 1)

define({CREGFU0_RBUS0}, 1)
define({CREGFU0_RBUS1}, 1)
```

```
define({CREGFU0_RBUS2}, 0)
define({CREGFU0_RBUS3}, 0)

define({CREGFU1_TBUS0}, 0)
define({CREGFU1_TBUS1}, 0)
define({CREGFU1_TBUS2}, 1)
define({CREGFU1_TBUS3}, 1)

define({CREGFU1_RBUS0}, 1)
define({CREGFU1_RBUS1}, 1)
define({CREGFU1_RBUS2}, 0)
define({CREGFU1_RBUS3}, 0)

define({CREGFU2_TBUS0}, 1)
define({CREGFU2_TBUS1}, 0)
define({CREGFU2_TBUS2}, 1)
define({CREGFU2_TBUS3}, 0)

define({CREGFU2_RBUS0}, 0)
define({CREGFU2_RBUS1}, 1)
define({CREGFU2_RBUS2}, 0)
define({CREGFU2_RBUS3}, 1)

define({CREGFU3_TBUS0}, 1)
define({CREGFU3_TBUS1}, 0)
define({CREGFU3_TBUS2}, 1)
define({CREGFU3_TBUS3}, 0)

define({CREGFU3_RBUS0}, 0)
define({CREGFU3_RBUS1}, 1)
define({CREGFU3_RBUS2}, 0)
define({CREGFU3_RBUS3}, 1)

define({CREGFU4_TBUS0}, 0)
define({CREGFU4_TBUS1}, 1)
define({CREGFU4_TBUS2}, 0)
define({CREGFU4_TBUS3}, 1)

define({CREGFU4_RBUS0}, 1)
define({CREGFU4_RBUS1}, 0)
define({CREGFU4_RBUS2}, 1)
define({CREGFU4_RBUS3}, 0)

define({CREGFU5_TBUS0}, 0)
define({CREGFU5_TBUS1}, 1)
define({CREGFU5_TBUS2}, 0)
define({CREGFU5_TBUS3}, 1)

define({CREGFU5_RBUS0}, 1)
define({CREGFU5_RBUS1}, 0)
define({CREGFU5_RBUS2}, 1)
define({CREGFU5_RBUS3}, 0)

define({CREGFU6_TBUS0}, 1)
define({CREGFU6_TBUS1}, 0)
define({CREGFU6_TBUS2}, 0)
define({CREGFU6_TBUS3}, 1)

define({CREGFU6_RBUS0}, 0)
define({CREGFU6_RBUS1}, 1)
define({CREGFU6_RBUS2}, 1)
define({CREGFU6_RBUS3}, 0)

define({CREGFU7_TBUS0}, 1)
```

```
define({CREGFU7_TBUS1}, 0)
define({CREGFU7_TBUS2}, 0)
define({CREGFU7_TBUS3}, 1)

define({CREGFU7_RBUS0}, 0)
define({CREGFU7_RBUS1}, 1)
define({CREGFU7_RBUS2}, 1)
define({CREGFU7_RBUS3}, 0)

define({CREGFU8_TBUS0}, 0)
define({CREGFU8_TBUS1}, 1)
define({CREGFU8_TBUS2}, 1)
define({CREGFU8_TBUS3}, 0)

define({CREGFU8_RBUS0}, 1)
define({CREGFU8_RBUS1}, 0)
define({CREGFU8_RBUS2}, 0)
define({CREGFU8_RBUS3}, 1)

define({CREGFU9_TBUS0}, 0)
define({CREGFU9_TBUS1}, 1)
define({CREGFU9_TBUS2}, 1)
define({CREGFU9_TBUS3}, 0)

define({CREGFU9_RBUS0}, 1)
define({CREGFU9_RBUS1}, 0)
define({CREGFU9_RBUS2}, 0)
define({CREGFU9_RBUS3}, 1)
```

C

VHDL description files

This appendix contains all the parametrised VHDL templates from which the VHDL processor description is generated. They may appear difficult to read at first because of the sometimes complicated preprocessor commands. It may be helpful to compare the sources with the resulting description file after preprocessing to gain some insight in the workings of these commands. Each file contains a header which describes the entity it contains and the architecture parameters on which this entity depends.

```
-----------------------------------------------------------------------------
-- Ifetch FU cache control
--
-- parameter dependencies: none
-----------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use work.ifetchcontrol.all;

entity cchctl is
   port(
insvalid : in std_logic; -- instruction valid (hit detected or
-- read complete (DC stage signal)
rwn : out std_logic; -- cache RAM 0=read/1=write
cwrite : out std_logic; -- cache write
ias : out std_logic; -- instruction address strobe
il : out std_logic; -- instruction latch
insext : out std_logic; -- instruction reg select external data
insen : out std_logic; -- instruction reg enable
insenforce : out std_logic; -- force instruction reg enable
glockreq : out std_logic;
glock : in std_logic;
rst : in std_logic;
ck : in std_logic);
end cchctl;


architecture func of cchctl is
   signal intern_read, intern_insenforce: std_logic;
   for fsm:cchfsm use entity work.cchfsm(func);
```

```
begin
   fsm:cchfsm
      port map(
insvalid => insvalid,
rst => rst,
rwn => rwn,
cwrite => cwrite,
read => intern_read,
il => il,
insext => insext,
insenforce => intern_insenforce,
ck => ck);
   ias <= intern_read and not insvalid;
   insen <= intern_insenforce or not glock;
   insenforce <= intern_insenforce;
   glockreq <= not intern_read or not insvalid;
end func;
------------------------------------------------------------------------------
-- Ifetch FU cache finite state machine
--
--`{ parameter dependencies: IFEFUDELAY }'
------------------------------------------------------------------------------

library IEEE;
use IEEE.Std_Logic_1164.all;

entity cchfsm is
   port(
insvalid : in std_logic;
rst : in std_logic;
rwn : out std_logic;
cwrite : out std_logic;
read : out std_logic;
il : out std_logic;
insext : out std_logic;
insenforce : out std_logic;
ck : in std_logic);
end cchfsm;


architecture func of cchfsm is
   type cache_states is (st_read, FOR(COUNT,2,IFEFUDELAY-1,st_delay`'COUNT`, ')
 st_write, st_recover);
begin
   process
      variable state : cache_states;
   begin
      wait until ck'event and ck = '1';
      if not(rst='1') then
 state := st_read;
      else
       case state is
when st_read =>
 rwn <= '1';
 cwrite <= '0';
 il <= '0';
 insext <= '0';
 insenforce <= '0';
 read <= '1';
 if not(insvalid = '1') then
    ifelse(eval(IFEFUDELAY>2), 1, state := st_delay2;, state := st_write;)
 end if;

FOR(COUNT,2,IFEFUDELAY-2,
{ }when `st_delay'COUNT =>
 rwn <= '1';
 cwrite <= '1';
 il <= '0';
 insext <= '0';
 insenforce <= '0';
 read <= '0';
 state := `st_delay'_eval(COUNT+1);
)
ifelse(eval(IFEFUDELAY>2), 1,
{ }when `st_delay'eval(IFEFUDELAY-1) =>
 rwn <= '1';
 cwrite <= '1';
 il <= '0';
 insext <= '0';
 insenforce <= '0';
 read <= '0';
 state := st_write;)

when st_write =>
 rwn <= '0';
 cwrite <= '1';
 il <= '1';
```

```
 insext <= '1';
 insenforce <= '1';
 read <= '0';
 state := st_recover;

when st_recover =>
 rwn <= '1';
 cwrite <= '0';
 il <= '0';
 insext <= '0';
 insenforce <= '0';
 read <= '0';
 state := st_read;
      end case;
     end if;
   end process;
end func;
------------------------------------------------------------------------------
-- Ifetch FU cache path
--
--`{ parameter dependencies:  IFECACHERAMASYN IFECACHESIZE IFEADDRSIZE}'
------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity cchpth is
   port(
ifdef({IFECACHERAMASYN},,
{ newifca : in std_logic_vector(LOG(eval(IFECACHESIZE-1))-1 downto 0);
-- new IF stage cache address.
}) ifa : in std_logic_vector(IFEADDRSIZE-1 downto 0);  -- IF stage address.
dca : in std_logic_vector(IFEADDRSIZE-1 downto 0);  -- DC stage address.

cwrite : in std_logic; -- cache write.

rama : out std_logic_vector(LOG(eval(IFECACHESIZE-1))-1 downto 0);
-- cache RAM address.
dt : out std_logic_vector(IFETAGSIZE-1 downto 0);
-- to cache RAM tag input.

qt : in std_logic_vector(IFETAGSIZE-1 downto 0);
-- to cache RAM tag output.
cenabled : in std_logic; -- cache (hit & read) enabled.
insenforce : in std_logic; -- force insvalid.
insen : in std_logic; -- insvalid latch enable.
insvalid : out std_logic; -- instr. (DC) register valid.

iin : in std_logic_vector(INSTRSIZE-1 downto 0); -- instr. from outside.
inscache : in std_logic_vector(INSTRSIZE-1 downto 0); -- instr. from cch.
insreg : out std_logic_vector(INSTRSIZE-1 downto 0); -- instr. out

insext : in std_logic; -- sel. instr. from cache/outs.

glock : in std_logic;
rst : in std_logic;
ck : in std_logic);
end cchpth;


architecture func of cchpth is
   signal hit, insvalid1 : boolean;
begin

   process(cwrite, glock, newifca, ifa, dca)
   begin
      if not(cwrite = '1') then -- switch stages
         if (glock='1') then -- second ugliest data hold ever.
    rama <= ifa(LOG(eval(IFECACHESIZE-1))-1 downto 0);
         else
    rama <= newifca;
         end if;
      else
         rama <= dca(LOG(eval(IFECACHESIZE-1))-1 downto 0); --one cycle extra delay
      end if;
   end process;

   insreg <= iin when (insext = '1') else
     inscache;

   dt <= dca(IFEADDRSIZE-1 downto LOG(eval(IFECACHESIZE-1)));
   hit <= (qt = ifa(IFEADDRSIZE-1 downto LOG(eval(IFECACHESIZE-1))) );

   process
   begin
      wait until ck'event and ck='1';
      if not(rst='1') then
```

```
        insvalid1 <= FALSE;
      elsif (insen='1') then
        insvalid1 <= (hit and (cenabled='1')) or (insenforce='1');
      end if;
   end process;

   insvalid <= '1' when insvalid1 else '0';
end func;
-------------------------------------------------------------------------------
-- Compare FU FUNR
--
--`{ parameter dependencies: PARMOVES BUSWIDTH CCMPFU}'FUNR...
-------------------------------------------------------------------------------

library IEEE;
use IEEE.Std_Logic_1164.all;

entity cmpfu`'FUNR is
   port(
FOR(BUSNR, 0, PARMOVES-1,
--
_ifelse(_eval((CCMPFU`'FUNR`'_TBUS`'BUSNR || CCMPFU`'FUNR`'_OBUS`'BUSNR) &&
CCMPFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : inout std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval((CCMPFU`'FUNR`'_TBUS`'BUSNR || CCMPFU`'FUNR`'_OBUS`'BUSNR) &&
!CCMPFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval(!(CCMPFU`'FUNR`'_TBUS`'BUSNR || CCMPFU`'FUNR`'_OBUS`'BUSNR) &&
CCMPFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);)
_ifelse(_eval(CCMPFU`'FUNR`'_TBUS`'BUSNR || CCMPFU`'FUNR`'_OBUS`'BUSNR ||
CCMPFU`'FUNR`'_RBUS`'BUSNR), 1,
squash`'BUSNR : in std_logic;)
_ifelse(_eval(CCMPFU`'FUNR`'_TBUS`'BUSNR || CCMPFU`'FUNR`'_OBUS`'BUSNR), 1,
dstbus`'BUSNR : in std_logic_vector(DSTSIZE`'BUSNR-1 downto 0);)
_ifelse( CCMPFU`'FUNR`'_RBUS`'BUSNR, 1,
srcbus`'BUSNR : in std_logic_vector(SRCSIZE`'BUSNR-1 downto 0);)
)--
texcreq : out std_logic; -- trigger socket exception request
oexcreq : out std_logic; -- operand socket exception request

squall : in std_logic;
glock : in std_logic;
glockreq : out std_logic;
sel : in std_logic_vector(eval(PARMOVES-1) downto 0);
rout : out std_logic;
outvalid : out std_logic; -- valid std_logic output
valen : in std_logic; -- enable valid std_logic

ck : in std_logic;
rst : in std_logic
   );
end cmpfu`'FUNR;

use work.sockets.all;
use work.internals.all;

architecture func of cmpfu`'FUNR is
define({NRTBUSES}, translit(NCMPFU\FUNR\_T,\))
define({NROBUSES}, translit(NCMPFU\FUNR\_O,\))
define({NRRBUSES}, translit(NCMPFU\FUNR\_R,\))

   for tsock:insknb`'NRTBUSES use entity WORK.insknb`'NRTBUSES`'(ARCHSTYLE);
   for osock:inspnb`'NROBUSES use entity WORK.inspnb`'NROBUSES`'(ARCHSTYLE);
   for rsock:outreg`'NRRBUSES use entity WORK.outreg`'NRRBUSES`'(ARCHSTYLE);
   for regsock:outcmp use entity WORK.outcmp(ARCHSTYLE);
   for fu_internals:cmpfui use entity WORK.cmpfui(ARCHSTYLE);
   for valsock:outval use entity WORK.outval(ARCHSTYLE);

   signal intern_tdata : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_topcode : std_logic_vector(LOG(eval(CMPFUTS-1))-1 downto 0);
   signal intern_tcls : std_logic;
   signal intern_toutsels : std_logic;
   signal intern_tdataact : std_logic;
   signal intern_tsquash : std_logic_vector(NRTBUSES-1 downto 0);

   signal intern_odata : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_ocls : std_logic;
   signal intern_ooutsels : std_logic;
   signal intern_odataact : std_logic;
   signal intern_osquash : std_logic_vector(NROBUSES-1 downto 0);

   signal intern_rdata : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_ract : std_logic;
   signal intern_remp : std_logic;
   signal intern_rsocketcls : std_logic;
   signal intern_routsels : std_logic;
```

```
   signal intern_rsquash : std_logic_vector(NRRBUSES-1 downto 0);

   signal intern_rout : std_logic;

begin
   tsock:insknb`'NRTBUSES
define({BSUBST}, translit(BCMPFU\FUNR\_T,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NRTBUSES-1,
{ }ldstsize`'BUSNR => _sub(DSTSIZE`'BSUBST\BUSNR){,}
{ }lowid`'BUSNR => _sub(ACMPFU\FUNR\_TBUS`'BSUBST\BUSNR){,}
){ }logopcodes => LOG(eval(CMPFUTS-1))) ,\)
translit(
{      }port map(
FOR(BUSNR, 0, NRTBUSES-1,
{ }databus`'BUSNR => databus`'BSUBST\BUSNR{,}
{ }dstbus`'BUSNR => dstbus`'BSUBST\BUSNR{,}
){ }excreq => texcreq,
{ }squash => intern_tsquash,
{ }glock => glock,
{ }ck => ck,
{ }data => intern_tdata,
{ }opcode => intern_topcode,
{ }fucls => intern_tcls,
{ }outsels => intern_toutsels,
{ }dataact => intern_tdataact );,\)

translit(
FOR(BUSNR, 0, NRTBUSES-1, {   }intern_tsquash(BUSNR) <= squash`'BSUBST\BUSNR;
),\)

   osock:inspnb`'NROBUSES
define({BSUBST}, translit(BCMPFU\FUNR\_O,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NROBUSES-2,
{ }ldstsize`'BUSNR => _sub(DSTSIZE`'BSUBST\BUSNR){,}
{ }lowid`'BUSNR => _sub(ACMPFU\FUNR\_OBUS`'BSUBST\BUSNR){,}
){ }ldstsize`'eval(NROBUSES-1) =>_sub(DSTSIZE`'BSUBST\eval(NROBUSES-1)){,}
{ }lowid`'eval(NROBUSES-1) => _sub(ACMPFU\FUNR\_OBUS`'BSUBST\eval(NROBUSES-1))),\)
translit(
{      }port map(
FOR(BUSNR, 0, NROBUSES-1,
{ }databus`'BUSNR => databus`'BSUBST\BUSNR{,}
{ }dstbus`'BUSNR => dstbus`'BSUBST\BUSNR{,}
){ }excreq => oexcreq,
{ }squash => intern_osquash,
{ }glock => glock,
{ }ck => ck,
{ }data => intern_odata,
{ }fucls => intern_ocls,
{ }outsels => intern_ooutsels,
{ }dataact => intern_odataact );,\)

translit(
FOR(BUSNR, 0, NROBUSES-1, {   }intern_osquash(BUSNR) <= squash`'BSUBST\BUSNR;
),\)

   rsock:outreg`'NRRBUSES
define({BSUBST}, translit(BCMPFU\FUNR\_R,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NRRBUSES-2,
{ }lsrcsize`'BUSNR => _sub(SRCSIZE`'BSUBST\BUSNR){,}
{ }lowid`'BUSNR => _sub(ACMPFU\FUNR\_RBUS`'BSUBST\BUSNR){,}
){ }lsrcsize`'eval(NRRBUSES-1) =>_sub(SRCSIZE`'BSUBST\eval(NRRBUSES-1)){,}
{ }lowid`'eval(NRRBUSES-1) => _sub(ACMPFU\FUNR\_RBUS`'BSUBST\eval(NRRBUSES-1))),\)
translit(
{      }port map(
FOR(BUSNR, 0, NRRBUSES-1,
{ }databus`'BUSNR => databus`'BSUBST\BUSNR{,}
{ }srcbus`'BUSNR => srcbus`'BSUBST\BUSNR{,}
){ }glock => glock,
{ }ck => ck,
{ }data => intern_rdata);,\)

   regsock:outcmp
      port map(
rout => rout,
sel => sel,
squall => squall,
glockreq => glockreq,
glock => glock,
rin => intern_rout,
fuact => intern_ract,
fuemp => intern_remp,
```

```
socketcls => intern_rsocketcls,
outsels => intern_routsels );

   fu_internals:cmpfui
{      }port map(
{ }tdata => intern_tdata,
{ }topcode => intern_topcode,
{ }tcls => intern_tcls,
{ }toutsels => intern_toutsels,
{ }tdataact => intern_tdataact,

{ }odata => intern_odata,
{ }ocls => intern_ocls,
{ }ooutsels => intern_ooutsels,
{ }odataact => intern_odataact,

{ }rdata => intern_rdata,
{ }ract => intern_ract,
{ }remp => intern_remp,
{ }rsocketcls => intern_rsocketcls,
{ }routsels => intern_routsels,

{ }rout => intern_rout,

{ }ck => ck,
{ }rst => rst);

   valsock:outval
      port map(
outvalid => outvalid,
vin => intern_ract,
enable => valen);

end func;
-------------------------------------------------------------------------------
-- Compare FU combinational logic
--
--`{ parameter dependencies: BUSWIDTH}'
-------------------------------------------------------------------------------

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

entity cmpfuc is
   port(a : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
b : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
opc : in std_logic_vector(LOG(eval(CMPFUTS-1))-1 downto 0);
y : out std_logic);
end cmpfuc;

architecture func of cmpfuc is
begin
   process(a, b, opc)
   begin
      case opc is
 when "00" => if signed(a)=0 then y<='1'; else y<='0'; end if;
 when "01" => if signed(a)=signed(b) then y<='1'; else y<='0'; end if;
 when "10" => if signed(a)>0 then y<='1'; else y<='0'; end if;
 when "11" => if signed(a)<0 then y<='1'; else y<='0'; end if;
 when others => y<='-';
      end case;
   end process;
end func;
-------------------------------------------------------------------------------
-- Compare FU internals
--
--`{ parameter dependencies: BUSWIDTH}'
-------------------------------------------------------------------------------

library IEEE;
use IEEE.Std_Logic_1164.all;

entity cmpfui is
   port(
tdata : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
topcode : in std_logic_vector(LOG(eval(CMPFUTS-1))-1 downto 0);
tcls : out std_logic;
toutsels : out std_logic;
tdataact : in std_logic;

odata : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
ocls : out std_logic;
ooutsels : out std_logic;
odataact : in std_logic;

rdata : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
```

```
ract : out std_logic;
remp : out std_logic;
rsocketcls : in std_logic;
routsels : in std_logic;

rout : out std_logic;

ck : in std_logic;
rst : in std_logic);
end cmpfui;

use work.stagecontrol.all;
use work.combinational.all;

architecture func of cmpfui is

   for stagectl0:stagecnb use entity WORK.stagecnb(ARCHSTYLE);
   for stagectl1:stagecnb use entity WORK.stagecnb(ARCHSTYLE);
   for comblogic:cmpfuc use entity WORK.cmpfuc(ARCHSTYLE);

   signal intern_outcls : std_logic;
   signal intern_outactive : std_logic;
   signal intern_outempty : std_logic;
   signal intern_incls : std_logic;

   signal stg0_enable : std_logic;
   signal stg1_enable : std_logic;

   signal wired_to_one : std_logic;
   signal wired_to_zero : std_logic;

   signal treg : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal oreg : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal rreg : std_logic;
   signal opcreg : std_logic_vector(LOG(eval(CMPFUTS-1))-1 downto 0);

begin
   stagectl0:stagecnb
port map(inactive  => tdataact,
                 inempty   => wired_to_one,
 incls     => intern_incls,

 ck        => ck,
 rst       => rst,
 socketcls => rsocketcls,
 enable    => stg0_enable,

 outactive => intern_outactive,
 outempty  => intern_outempty,
 outcls    => intern_outcls);
   stagectl1:stagecnb
port map(inactive  => intern_outactive,
                 inempty   => intern_outempty,
 incls     => intern_outcls,

 ck        => ck,
 rst       => rst,
 socketcls => rsocketcls,
 enable    => stg1_enable,

 outactive => ract,
 outempty  => remp,
 outcls    => wired_to_one);
   comblogic:cmpfuc
port map(a => treg,
 b => oreg,
 opc => opcreg,
 y => rreg);

   wired_to_one <= '1';
   wired_to_zero <= '0';

   tcls <= intern_incls;
   ocls <= intern_incls;
   toutsels <= routsels;
   ooutsels <= routsels;

   process
   begin
      wait on ck until ck='1';
      if (stg1_enable='1') then
 rout <= rreg;
      end if;
      if (stg0_enable='1') then
 treg <= tdata;
 opcreg <= topcode;
      end if;
```

```
      if (odataact='1') then
 oreg <= odata;
      end if;
   end process;

   rdata <= oreg;

end func;
-------------------------------------------------------------------------------
--  Global Interconnect
--
--`{ parameter dependencies: PARMOVES BUSWIDTH IFEADDRSIZE LDSADDRSIZE LDSFUS
--   RFLFUS MULFUS INTFUS LOGFUS SHIFUS CMPFUS IMMFUS GUARDSPECSIZE CxxFUxx.}'
-------------------------------------------------------------------------------
--
--
--  The globic system specifies the global level of interconnect:
--  move data, ID, and control busses, plus some specific connections
--  between FU's. For details on how preprocessing is used see
--  parmderv.sih and intfu.sih.
--
--  Besides FUs, the instruction cache RAMs are placed on this level
--  too.
--

library IEEE;
use IEEE.std_logic_1164.all;

entity globic is
   port(iin : in std_logic_vector(INSTRSIZE-1 downto 0);
ia : out std_logic_vector(IFEADDRSIZE-1 downto 0);
ias : out std_logic;
il : out std_logic;
int : in std_logic;
FOR(CNT, 0, LDSFUS-1,
{ }dout`'CNT : out std_logic_vector(BUSWIDTH-1 downto 0);
{ }din`'CNT : in std_logic_vector(BUSWIDTH-1 downto 0);
{ }wnr`'CNT : out std_logic;
{ }da`'CNT : out std_logic_vector(LDSADDRSIZE-1 downto 0);
{ }das`'CNT : out std_logic;
{ }dl`'CNT : out std_logic;)
ck : in std_logic;
rst : in std_logic);
end globic;

use work.funcunits.all;

architecture func of globic is

   for ife0 : ifefu0 use entity work.ifefu0(ARCHSTYLE);
FOR(CNT,0,RFLFUS-1,{
   }for rfl`'CNT : rflfu`'CNT use entity work.rflfu`'CNT`'(ARCHSTYLE);)

FOR(CNT,0,MULFUS-1,{
   }for mul`'CNT : mulfu`'CNT use entity work.mulfu`'CNT`'(ARCHSTYLE);)

FOR(CNT,0,INTFUS-1,{
   }for int`'CNT : intfu`'CNT use entity work.intfu`'CNT`'(ARCHSTYLE);)

FOR(CNT,0,LOGFUS-1,{
   }for log`'CNT : logfu`'CNT use entity work.logfu`'CNT`'(ARCHSTYLE);)

FOR(CNT,0,SHIFUS-1,{
   }for shi`'CNT : shifu`'CNT use entity work.shifu`'CNT`'(ARCHSTYLE);)

FOR(CNT,0,CMPFUS-1,{
   }for cmp`'CNT : cmpfu`'CNT use entity work.cmpfu`'CNT`'(ARCHSTYLE);)

FOR(CNT,0,LDSFUS-1,{
   }for lds`'CNT : ldsfu`'CNT use entity work.ldsfu`'CNT`'(ARCHSTYLE);)

FOR(CNT,0,IMMFUS-1,{
   }for imm`'CNT : immfu`'CNT use entity work.immfu`'CNT`'(ARCHSTYLE);)

FOR(CNT,0,REGFUS-1,{
   }for reg`'CNT : regfu`'CNT use entity work.regfu`'CNT`'(ARCHSTYLE);)

   for grd  : grdfu use entity work.grdfu(ARCHSTYLE);
   for val  : valfu use entity work.valfu(ARCHSTYLE);
   for net  : netcontrol use entity work.netcontrol(ARCHSTYLE);

FOR(CNT,0,PARMOVES-1,{
   }signal sdatabus`'CNT : std_logic_vector(BUSWIDTH-1 downto 0);)

   -- Instruction cache bus mapping signals
   signal dall : std_logic_vector(eval(INSTRSIZE+IFETAGSIZE-1) downto 0);
   signal qall : std_logic_vector(eval(INSTRSIZE+IFETAGSIZE-1) downto 0);
```

```
FOR(CNT, 0, PARMOVES-1,
{   }signal grd_squash`'CNT : std_logic;
)   signal ife0_squall : std_logic;

FOR(NDX, 0, PARMOVES-1,
{   }signal ife0_srcbus`'NDX : std_logic_vector(_sub(SRCSIZE`'NDX)-1 downto 0);
{   }signal ife0_dstbus`'NDX : std_logic_vector(_sub(DSTSIZE`'NDX)-1 downto 0);
ifdef({SINGLEGUARD},,{   }signal grd_guardspec`'NDX : std_logic_vector(GUARDSPECSIZE-1 downto 0);)
)
ifdef({SINGLEGUARD},{   }signal grd_guardspec : std_logic_vector(GUARDSPECSIZE-1 downto 0);)

FOR(NDX,0,CMPFUS-1,
{   }signal grd_cmpsel`'NDX : std_logic_vector(PARMOVES-1 downto 0);
{   }signal grd_cmprout`'NDX : std_logic;
)
   signal net_glock : std_logic;
   signal net_glockreq : std_logic;
   signal net_cmpglock : std_logic_vector(CMPFUS-1 downto 0);
FOR(BUSNR,0,PARMOVES-1,
_ifelse(_eval(`LLOCKREQS'BUSNR >= 0), 1, `
   'signal net_llockreq`'BUSNR : std_logic_vector(`LLOCKREQS'BUSNR downto 0);))
   signal net_squash : std_logic_vector(PARMOVES-1 downto 0);
   signal net_excreq : std_logic_vector(EXCREQS downto 0);
   signal net_exc : std_logic;

   signal val_valen : std_logic_vector(NVALFU_R-1 downto 0);
begin
define({FUNR},0)
   ife0 : ifefu0
      port map(

FOR(NDX, 0, PARMOVES-1,
_ifelse(_eval(CIFEFU`'FUNR`'_TBUS`'NDX || CIFEFU`'FUNR`'_R1BUS`'NDX ||
        CIFEFU`'FUNR`'_R2BUS`'NDX), 1,
{ }databus`'NDX => sdatabus`'NDX`,')

_ifelse(_eval(CIFEFU`'FUNR`'_TBUS`'NDX || CIFEFU`'FUNR`'_R2BUS`'NDX), 1,
{ }squash`'NDX => grd_squash`'NDX`,')
)

FOR(NDX, 0, PARMOVES-1,
{ }srcbus`'NDX => ife0_srcbus`'NDX{,}
{ }dstbus`'NDX => ife0_dstbus`'NDX{,}
{ifdef({SINGLEGUARD},,{ }guardspec`'NDX => grd_guardspec`'NDX{,})}
)
ifdef({SINGLEGUARD},{ }guardspec => grd_guardspec{,})

{ }squall => ife0_squall,
{ }iin => iin,
{ }ia => ia,
{ }ias => ias,
{ }il => il,
{ }int => int,
{ }glock => net_glock,
{ }glockreq => net_glockreq,
{ }exc => net_exc,
{ }ck => ck,
{ }rst => rst);

FOR(LOOPFUCNT, 0, RFLFUS-1,
define({FUNR}, LOOPFUCNT){
{   }rfl`'FUNR : rflfu`'FUNR
{      }port map(
FOR(NDX, 0, PARMOVES-1,
_ifelse(_eval(CRFLFU`'FUNR`'_TBUS`'NDX || CRFLFU`'FUNR`'_RBUS`'NDX), 1,
{ }databus`'NDX => sdatabus`'NDX`,')

_ifelse( CRFLFU`'FUNR`'_TBUS`'NDX, 1,
{ }squash`'NDX => grd_squash`'NDX`,')

_ifelse( CRFLFU`'FUNR`'_TBUS`'NDX, 1,
{ }dstbus`'NDX => ife0_dstbus`'NDX`,')

_ifelse( CRFLFU`'FUNR`'_RBUS`'NDX, 1,
{ }srcbus`'NDX => ife0_srcbus`'NDX`,')
)
glock => net_glock{,}
ck => ck);
})

FOR(LOOPFUCNT, 0, MULFUS-1,
define({FUNR}, LOOPFUCNT){
{   }mul`'FUNR : mulfu`'FUNR
{      }port map(
FOR(BUSNR, 0, PARMOVES-1,
_ifelse(_eval(CMULFU`'FUNR`'_TBUS`'BUSNR || CMULFU`'FUNR`'_OBUS`'BUSNR ||
```

```
CMULFU`'FUNR`'_R1BUS`'BUSNR || CMULFU`'FUNR`'_R2BUS`'BUSNR), 1,
{ }databus`'BUSNR => sdatabus`'BUSNR`',''
{ }squash`'BUSNR => grd_squash`'BUSNR`',''

_ifelse(_eval(CMULFU`'FUNR`'_TBUS`'BUSNR || CMULFU`'FUNR`'_OBUS`'BUSNR), 1,
{ }dstbus`'BUSNR => ife0_dstbus`'BUSNR`','')

_ifelse(_eval(CMULFU`'FUNR`'_R1BUS`'BUSNR || CMULFU`'FUNR`'_R2BUS`'BUSNR), 1,
{ }srcbus`'BUSNR => ife0_srcbus`'BUSNR`','')

_ifelse(CMULFU`'FUNR`'_TBUS`'BUSNR, 1,
{ }tllockreq`'BUSNR => net_llockreq`'BUSNR`'(SMULFU`'FUNR`'_TLLOCKREQ`'BUSNR)`','')

_ifelse(CMULFU`'FUNR`'_OBUS`'BUSNR, 1,
{ }ollockreq`'BUSNR => net_llockreq`'BUSNR`'(SMULFU`'FUNR`'_OLLOCKREQ`'BUSNR)`','')

_ifelse(CMULFU`'FUNR`'_R1BUS`'BUSNR, 1,
{ }r1llockreq`'BUSNR => net_llockreq`'BUSNR`'(SMULFU`'FUNR`'_R1LLOCKREQ`'BUSNR)`','')

_ifelse(CMULFU`'FUNR`'_R2BUS`'BUSNR, 1,
{ }r2llockreq`'BUSNR => net_llockreq`'BUSNR`'(SMULFU`'FUNR`'_R2LLOCKREQ`'BUSNR)`','')
)
texcreq => net_excreq(_sub(SMULFU`'FUNR`'_TEXCREQ)){,}
oexcreq => net_excreq(_sub(SMULFU`'FUNR`'_OEXCREQ)){,}
r1excreq => net_excreq(_sub(SMULFU`'FUNR`'_R1EXCREQ)){,}
r2excreq => net_excreq(_sub(SMULFU`'FUNR`'_R2EXCREQ)){,}
glock => net_glock{,}
outvalid => sdatabus`'_sub(BVALFU_R`'_sub(SMULFU`'FUNR`'_VALREG))`'(_sub(SMULFU`'FUNR`'_VALBIT)){,}
valen => val_valen(_sub(SMULFU`'FUNR`'_VALREG)){,}
ck => ck{,}
rst => rst);
})

FOR(LOOPFUCNT, 0, INTFUS-1,
define({FUNR}, LOOPFUCNT){
{   }int`'FUNR : intfu`'FUNR
{      }port map(
FOR(BUSNR, 0, PARMOVES-1,
_ifelse(_eval(CINTFU`'FUNR`'_TBUS`'BUSNR || CINTFU`'FUNR`'_OBUS`'BUSNR ||
CINTFU`'FUNR`'_RBUS`'BUSNR), 1,
{ }databus`'BUSNR => sdatabus`'BUSNR`',''
{ }squash`'BUSNR => grd_squash`'BUSNR`',''

_ifelse(_eval(CINTFU`'FUNR`'_TBUS`'BUSNR || CINTFU`'FUNR`'_OBUS`'BUSNR), 1,
{ }dstbus`'BUSNR => ife0_dstbus`'BUSNR`','')

_ifelse(CINTFU`'FUNR`'_RBUS`'BUSNR, 1,
{ }srcbus`'BUSNR => ife0_srcbus`'BUSNR`','')

_ifelse(CINTFU`'FUNR`'_TBUS`'BUSNR, 1,
{ }tllockreq`'BUSNR => net_llockreq`'BUSNR`'(SINTFU`'FUNR`'_TLLOCKREQ`'BUSNR)`','')

_ifelse(CINTFU`'FUNR`'_OBUS`'BUSNR, 1,
{ }ollockreq`'BUSNR => net_llockreq`'BUSNR`'(SINTFU`'FUNR`'_OLLOCKREQ`'BUSNR)`','')

_ifelse(CINTFU`'FUNR`'_RBUS`'BUSNR, 1,
{ }rllockreq`'BUSNR => net_llockreq`'BUSNR`'(SINTFU`'FUNR`'_RLLOCKREQ`'BUSNR)`','')
)
texcreq => net_excreq(_sub(SINTFU`'FUNR`'_TEXCREQ)){,}
oexcreq => net_excreq(_sub(SINTFU`'FUNR`'_OEXCREQ)){,}
rexcreq => net_excreq(_sub(SINTFU`'FUNR`'_REXCREQ)){,}
glock => net_glock{,}
outvalid => sdatabus`'_sub(BVALFU_R`'_sub(SINTFU`'FUNR`'_VALREG))`'(_sub(SINTFU`'FUNR`'_VALBIT)){,}
valen => val_valen(_sub(SINTFU`'FUNR`'_VALREG)){,}
ck => ck{,}
rst => rst);
})

FOR(LOOPFUCNT, 0, LOGFUS-1,
define({FUNR}, LOOPFUCNT){
{   }log`'FUNR : logfu`'FUNR
{      }port map(
FOR(BUSNR, 0, PARMOVES-1,
_ifelse(_eval(CLOGFU`'FUNR`'_TBUS`'BUSNR || CLOGFU`'FUNR`'_OBUS`'BUSNR ||
CLOGFU`'FUNR`'_RBUS`'BUSNR), 1,
{ }databus`'BUSNR => sdatabus`'BUSNR`',''
{ }squash`'BUSNR => grd_squash`'BUSNR`',''

_ifelse(_eval(CLOGFU`'FUNR`'_TBUS`'BUSNR || CLOGFU`'FUNR`'_OBUS`'BUSNR), 1,
{ }dstbus`'BUSNR => ife0_dstbus`'BUSNR`','')

_ifelse(CLOGFU`'FUNR`'_RBUS`'BUSNR, 1,
{ }srcbus`'BUSNR => ife0_srcbus`'BUSNR`','')

_ifelse(CLOGFU`'FUNR`'_RBUS`'BUSNR, 1,
{ }rllockreq`'BUSNR => net_llockreq`'BUSNR`'(SLOGFU`'FUNR`'_RLLOCKREQ`'BUSNR)`','')
)
```

```
texcreq => net_excreq(_sub(SLOGFU`'FUNR`'_TEXCREQ)){,}
oexcreq => net_excreq(_sub(SLOGFU`'FUNR`'_OEXCREQ)){,}
rexcreq => net_excreq(_sub(SLOGFU`'FUNR`'_REXCREQ)){,}
glock => net_glock{,}
outvalid => sdatabus`'_sub(BVALFU_R`'_sub(SLOGFU`'FUNR`'_VALREG))`'(_sub(SLOGFU`'FUNR`'_VALBIT)){,}
valen => val_valen(_sub(SLOGFU`'FUNR`'_VALREG)){,}
ck => ck{,}
rst => rst);
})

FOR(LOOPFUCNT, 0, SHIFUS-1,
define({FUNR}, LOOPFUCNT){
{   }shi`'FUNR : shifu`'FUNR
{      }port map(
FOR(BUSNR, 0, PARMOVES-1,
_ifelse(_eval(CSHIFU`'FUNR`'_TBUS`'BUSNR || CSHIFU`'FUNR`'_R1BUS`'BUSNR ||
CSHIFU`'FUNR`'_R2BUS`'BUSNR), 1,
{ }databus`'BUSNR => sdatabus`'BUSNR`',''
{ }squash`'BUSNR => grd_squash`'BUSNR`',''

_ifelse(CSHIFU`'FUNR`'_TBUS`'BUSNR, 1,
{ }dstbus`'BUSNR => ife0_dstbus`'BUSNR`',''

_ifelse(_eval(CSHIFU`'FUNR`'_R1BUS`'BUSNR || CSHIFU`'FUNR`'_R2BUS`'BUSNR), 1,
{ }srcbus`'BUSNR => ife0_srcbus`'BUSNR`',''

_ifelse(CSHIFU`'FUNR`'_R1BUS`'BUSNR, 1,
{ }r1llockreq`'BUSNR => net_llockreq`'BUSNR`'(SSHIFU`'FUNR`'_R1LLOCKREQ`'BUSNR)`',''

_ifelse(CSHIFU`'FUNR`'_R2BUS`'BUSNR, 1,
{ }r2llockreq`'BUSNR => net_llockreq`'BUSNR`'(SSHIFU`'FUNR`'_R2LLOCKREQ`'BUSNR)`',''
)
texcreq => net_excreq(_sub(SSHIFU`'FUNR`'_TEXCREQ)){,}
r1excreq => net_excreq(_sub(SSHIFU`'FUNR`'_R1EXCREQ)){,}
r2excreq => net_excreq(_sub(SSHIFU`'FUNR`'_R2EXCREQ)){,}
glock => net_glock{,}
outvalid => sdatabus`'_sub(BVALFU_R`'_sub(SSHIFU`'FUNR`'_VALREG))`'(_sub(SSHIFU`'FUNR`'_VALBIT)){,}
valen => val_valen(_sub(SSHIFU`'FUNR`'_VALREG)){,}
ck => ck{,}
rst => rst);
})

FOR(LOOPFUCNT, 0, CMPFUS-1,
define({FUNR}, LOOPFUCNT){
{   }cmp`'FUNR : cmpfu`'FUNR
{      }port map(
FOR(BUSNR, 0, PARMOVES-1,
_ifelse(_eval(CCMPFU`'FUNR`'_TBUS`'BUSNR || CCMPFU`'FUNR`'_OBUS`'BUSNR ||
CCMPFU`'FUNR`'_RBUS`'BUSNR), 1,
{ }databus`'BUSNR => sdatabus`'BUSNR`',''
{ }squash`'BUSNR => grd_squash`'BUSNR`',''

_ifelse(_eval(CCMPFU`'FUNR`'_TBUS`'BUSNR || CCMPFU`'FUNR`'_OBUS`'BUSNR), 1,
{ }dstbus`'BUSNR => ife0_dstbus`'BUSNR`',''

_ifelse( CCMPFU`'FUNR`'_RBUS`'BUSNR, 1,
{ }srcbus`'BUSNR => ife0_srcbus`'BUSNR`',''
)
texcreq => net_excreq(_sub(SCMPFU`'FUNR`'_TEXCREQ)){,}
oexcreq => net_excreq(_sub(SCMPFU`'FUNR`'_OEXCREQ)){,}
squall => ife0_squall{,}
glock => net_glock{,}
glockreq => net_cmpglock(FUNR){,}
sel => grd_cmpsel`'FUNR{,}
rout => grd_cmprout`'FUNR{,}
outvalid => sdatabus`'_sub(BVALFU_R`'_sub(SCMPFU`'FUNR`'_VALREG))`'(_sub(SCMPFU`'FUNR`'_VALBIT)){,}
valen => val_valen(_sub(SCMPFU`'FUNR`'_VALREG)){,}
ck => ck{,}
rst => rst);
})

FOR(LOOPFUCNT, 0, LDSFUS-1,
define({FUNR}, LOOPFUCNT){
{   }lds`'FUNR : ldsfu`'FUNR
{      }port map(
FOR(BUSNR, 0, PARMOVES-1,
_ifelse(_eval(CLDSFU`'FUNR`'_TBUS`'BUSNR || CLDSFU`'FUNR`'_OBUS`'BUSNR ||
CLDSFU`'FUNR`'_RBUS`'BUSNR), 1,
{ }databus`'BUSNR => sdatabus`'BUSNR`',''
{ }squash`'BUSNR => grd_squash`'BUSNR`',''

_ifelse(_eval(CLDSFU`'FUNR`'_TBUS`'BUSNR || CLDSFU`'FUNR`'_OBUS`'BUSNR), 1,
{ }dstbus`'BUSNR => ife0_dstbus`'BUSNR`',''

_ifelse( CLDSFU`'FUNR`'_RBUS`'BUSNR, 1,
{ }srcbus`'BUSNR => ife0_srcbus`'BUSNR`',''
```

```
_ifelse(CLDSFU`'FUNR`'_TBUS`'BUSNR, 1,
{ }tllockreq`'BUSNR => net_llockreq`'BUSNR`'(SLDSFU`'FUNR`'_TLLOCKREQ`'BUSNR)`',`')

_ifelse(CLDSFU`'FUNR`'_OBUS`'BUSNR, 1,
{ }ollockreq`'BUSNR => net_llockreq`'BUSNR`'(SLDSFU`'FUNR`'_OLLOCKREQ`'BUSNR)`',`')

_ifelse(CLDSFU`'FUNR`'_RBUS`'BUSNR, 1,
{ }rllockreq`'BUSNR => net_llockreq`'BUSNR`'(SLDSFU`'FUNR`'_RLLOCKREQ`'BUSNR)`',`')
)
rexcreq => net_excreq(_sub(SLDSFU`'FUNR`'_REXCREQ)){,}
glock => net_glock{,}
outvalid => sdatabus`'_sub(BVALFU_R`'_sub(SLDSFU`'FUNR`'_VALREG))`'(_sub(SLDSFU`'FUNR`'_VALBIT)){,}
valen => val_valen(_sub(SLDSFU`'FUNR`'_VALREG)){,}
dout => dout`'FUNR{,}
din => din`'FUNR{,}
wnr => wnr`'FUNR{,}
da => da`'FUNR{,}
das => das`'FUNR{,}
dl => dl`'FUNR{,}
ck => ck{,}
rst => rst);
})

FOR(LOOPFUCNT, 0, IMMFUS-1,
define({FUNR}, LOOPFUCNT){
{   }imm`'FUNR : immfu`'FUNR
{      }port map(
FOR(BUSNR, 0, PARMOVES-1,
_ifelse(CIMMFU`'FUNR`'_RBUS`'BUSNR, 1,
{ }databus`'BUSNR => sdatabus`'BUSNR`',`'
{ }srcbus`'BUSNR => ife0_srcbus`'BUSNR`',`')
)
glock => net_glock{,}
ck => ck);
})

FOR(LOOPFUCNT, 0, REGFUS-1,
define({FUNR}, LOOPFUCNT){
{   }reg`'FUNR : regfu`'FUNR
{      }port map(
FOR(BUSNR, 0, PARMOVES-1,
_ifelse(_eval(CREGFU`'FUNR`'_TBUS`'BUSNR || CREGFU`'FUNR`'_RBUS`'BUSNR), 1,
{ }databus`'BUSNR => sdatabus`'BUSNR`',`')

_ifelse( CREGFU`'FUNR`'_TBUS`'BUSNR, 1,
{ }squash`'BUSNR => grd_squash`'BUSNR`',`'
{ }dstbus`'BUSNR => ife0_dstbus`'BUSNR`',`')

_ifelse( CREGFU`'FUNR`'_RBUS`'BUSNR, 1,
{ }srcbus`'BUSNR => ife0_srcbus`'BUSNR`',`')
)
glock => net_glock{,}
ck => ck);
})

   grd : grdfu
      port map(
ifdef({SINGLEGUARD},
{ guardspec => grd_guardspec{,} },
{FOR(BUSNR,0,PARMOVES-1,
{ guardspec`'BUSNR => grd_guardspec`'BUSNR{,} }
)})
squall => ife0_squall{,}
FOR(BUSNR,0,CMPFUS-1,
{ }cmprout`'BUSNR => grd_cmprout`'BUSNR{,}
{ }cmpsel`'BUSNR => grd_cmpsel`'BUSNR{,}
)
FOR(BUSNR, 0, PARMOVES-1,
{ }squash`'BUSNR => grd_squash`'BUSNR{,}
)
glock => net_glock{,}
ck => ck);

   val : valfu
      port map(
FOR(BUSNR,0,PARMOVES-1,
_ifelse( CVALFU_RBUS`'BUSNR, 1,
{ srcbus`'BUSNR => ife0_srcbus`'BUSNR`',`'})
)
valen => val_valen{,}
glock => net_glock{,}
ck => ck);

   net : netcontrol
port map(
glockreq => net_glockreq,
cmpglock => net_cmpglock,
```

```
FOR(BUSNR, 0, PARMOVES-1,
{ }llockreq`'BUSNR => net_llockreq`'BUSNR{,}
)
squash => net_squash,
glock => net_glock,
excreq => net_excreq,
exc => net_exc);

FOR(BUSNR, 0, PARMOVES-1,{
   }net_squash(BUSNR) <= grd_squash`'BUSNR;)

end ARCHSTYLE;
------------------------------------------------------------------------------
-- Guard FU
--
--`{ parameter dependencies: PARMOVES BUSWIDTH SINGLEGUARD GUARDSPECSIZE}'
------------------------------------------------------------------------------

library IEEE;
use IEEE.Std_Logic_1164.all;

entity grdfu is
   port(
ifdef({SINGLEGUARD},,FOR(BUSNR, 0, PARMOVES-1,
`guardspec'BUSNR : in std_logic_vector(eval(GUARDSPECSIZE-1) downto 0);
))
ifdef({SINGLEGUARD},{ }guardspec : in std_logic_vector(eval(GUARDSPECSIZE-1) downto 0);)
squall : in std_logic;
FOR(BUSNR, 0, CMPFUS-1,
{ }cmprout`'BUSNR : in std_logic;
{ }cmpsel`'BUSNR : out std_logic_vector(PARMOVES-1 downto 0);
)
FOR(BUSNR, 0, PARMOVES-1,
{ }squash`'BUSNR : out std_logic;
)
glock : in std_logic;
ck : in std_logic);
end grdfu;


architecture func of grdfu is

ifdef({SINGLEGUARD},
{   signal guardspecl : std_logic_vector(GUARDSPECSIZE-1 downto 0);},
{FOR(BUSNR,0,PARMOVES-1,
{   }signal guardspecl`'BUSNR : std_logic_vector(GUARDSPECSIZE-1 downto 0);
)})

ifdef({SINGLEGUARD},
{   signal squashs : std_logic;
   signal cmpsels0 : std_logic;
   signal cmpsels1 : std_logic;},
{FOR(BUSNR,0,PARMOVES-1,
{   }signal squashs`'BUSNR : std_logic;
   signal cmpsels0`'BUSNR : std_logic;
   signal cmpsels1`'BUSNR : std_logic;
)})

begin
   process
   begin
      wait until ck'event and ck = '1';
      if not (glock = '1') then
ifdef({SINGLEGUARD},
{  guardspecl <= guardspec;
},{FOR(BUSNR,0,PARMOVES-1,
{  }guardspecl`'BUSNR <= guardspec`'BUSNR;
)})      end if;
   end process;

ifdef({SINGLEGUARD},
{   process(guardspecl, cmprout0, cmprout1)
   begin
      case guardspecl is
when "000" => squashs <= not cmprout0;
when "001" => squashs <= cmprout0;
when "010" => squashs <= not cmprout1;
when "011" => squashs <= cmprout1;
when "100" => squashs <= not (cmprout0 and cmprout1);
when "101" => squashs <= not (not cmprout0 and cmprout1);
when "110" => squashs <= not (cmprout0 and not cmprout1);
when "111" => squashs <= '0';
when others => squashs <= '-';
      end case;
   end process;
FOR(BUSNR,0,PARMOVES-1,
{   }squash`'BUSNR <= squashs`'BUSNR or squall;
```

```
)},

{FOR(BUSNR,0,PARMOVES-1,{
   process(guardspecl`'BUSNR, cmprout0, cmprout1)
   begin
      case guardspecl`'BUSNR is
when "000" => squashs`'BUSNR <= not cmprout0;
when "001" => squashs`'BUSNR <= cmprout0;
when "010" => squashs`'BUSNR <= not cmprout1;
when "011" => squashs`'BUSNR <= cmprout1;
when "100" => squashs`'BUSNR <= not (cmprout0 and cmprout1);
when "101" => squashs`'BUSNR <= not (not cmprout0 and cmprout1);
when "110" => squashs`'BUSNR <= not (cmprout0 and not cmprout1);
when "111" => squashs`'BUSNR <= '0';
when others => squashs`'BUSNR <= '-';
      end case;
   end process;
   squash`'BUSNR <= squashs`'BUSNR or squall;
})})

ifdef({SINGLEGUARD},
{   process(guardspecl)
   begin
      case guardspecl is
when "000" => cmpsels0 <= '1';
      cmpsels1 <= '0';
when "001" => cmpsels0 <= '1';
      cmpsels1 <= '0';
when "010" => cmpsels0 <= '0';
      cmpsels1 <= '1';
when "011" => cmpsels0 <= '0';
      cmpsels1 <= '1';
when "100" => cmpsels0 <= '1';
      cmpsels1 <= '1';
when "101" => cmpsels0 <= '1';
      cmpsels1 <= '1';
when "110" => cmpsels0 <= '1';
      cmpsels1 <= '1';
when "111" => cmpsels0 <= '0';
      cmpsels1 <= '0';
when others => cmpsels0 <= '-';
       cmpsels1 <= '-';
      end case;
   end process;
FOR(BUSNR,0,PARMOVES-1,
{   }cmpsel0(BUSNR) <= cmpsel0;
{   }cmpsel1(BUSNR) <= cmpsel1;
)
},
{FOR(BUSNR,0,PARMOVES-1,{
   process(guardspecl`'BUSNR)
   begin
      case guardspecl`'BUSNR is
when "000" => cmpsels0`'BUSNR <= '1';
      cmpsels1`'BUSNR <= '0';
when "001" => cmpsels0`'BUSNR <= '1';
      cmpsels1`'BUSNR <= '0';
when "010" => cmpsels0`'BUSNR <= '0';
      cmpsels1`'BUSNR <= '1';
when "011" => cmpsels0`'BUSNR <= '0';
      cmpsels1`'BUSNR <= '1';
when "100" => cmpsels0`'BUSNR <= '1';
      cmpsels1`'BUSNR <= '1';
when "101" => cmpsels0`'BUSNR <= '1';
      cmpsels1`'BUSNR <= '1';
when "110" => cmpsels0`'BUSNR <= '1';
      cmpsels1`'BUSNR <= '1';
when "111" => cmpsels0`'BUSNR <= '0';
      cmpsels1`'BUSNR <= '0';
when others => cmpsels0`'BUSNR <= '-';
       cmpsels1`'BUSNR <= '-';
      end case;
   end process;

   cmpsel0(BUSNR) <= cmpsels0`'BUSNR;
   cmpsel1(BUSNR) <= cmpsels1`'BUSNR;
})})
end func;
------------------------------------------------------------------------------
-- Instruction Cache RAM
------------------------------------------------------------------------------
library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Std_Logic_arith.all;

-- Dummy RAM cell entity
```

```
entity ramcell is
   generic(
addr_range : integer;
words : integer;
wordlength : integer);
   port(
ADD : in std_logic_vector(addr_range downto 0);
ME : in std_logic;
WE : in std_logic;
DI : in std_logic_vector(wordlength-1 downto 0);
DO : out std_logic_vector(wordlength-1 downto 0) );
end;

architecture func of ramcell is
begin
  -- This is a dummy architecture for a ramcell.
end;

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

entity icacheram is
   port(cache_addr : in std_logic_vector(eval(LOG(eval(IFECACHESIZE-1))-1) downto 0);
cache_datain : in std_logic_vector(eval(INSTRSIZE-1) downto 0);
cache_tagin : in std_logic_vector(eval(IFETAGSIZE-1) downto 0);

cache_dataout : out std_logic_vector(eval(INSTRSIZE-1) downto 0);
cache_tagout : out std_logic_vector(eval(IFETAGSIZE-1) downto 0);

cache_rwn : in std_logic ifdef({IFECACHERAMASYN},,{;
en : in std_logic})
);
end icacheram;

define({MAXRAMWORDSIZE}, eval(MAXRAMSIZE/IFECACHESIZE))
define({CACHEWORDSIZE}, eval(INSTRSIZE + IFETAGSIZE))
define({NUM_OF_MAXRAMS}, eval(CACHEWORDSIZE / MAXRAMWORDSIZE))
define({REMRAMWORDSIZE}, eval(CACHEWORDSIZE % MAXRAMWORDSIZE))

architecture func of icacheram is

 component ramcell
   generic(
addr_range : integer;
words : integer;
wordlength : integer);
   port(
ADD : in std_logic_vector(addr_range downto 0);
ME : in std_logic;
WE : in std_logic;
DI : in std_logic_vector(wordlength-1 downto 0);
DO : out std_logic_vector(wordlength-1 downto 0) );
 end component;

FOR(CHIPNR, 1, NUM_OF_MAXRAMS,
{   signal cache_datain_part`'CHIPNR : std_logic_vector(MAXRAMWORDSIZE-1 downto 0);
})
   signal cache_datain_partrest : std_logic_vector(REMRAMWORDSIZE-1 downto 0);

FOR(CHIPNR, 1, NUM_OF_MAXRAMS,
{   signal cache_dataout_part`'CHIPNR : std_logic_vector(MAXRAMWORDSIZE-1 downto 0);
})
   signal cache_dataout_partrest : std_logic_vector(REMRAMWORDSIZE-1 downto 0);

begin

FOR(CHIPNR, 1, NUM_OF_MAXRAMS,
{   ram`'CHIPNR : ramcell
      generic map(
addr_range => eval(LOG(eval(IFECACHESIZE-1))-1),
words => IFECACHESIZE,
wordlength => MAXRAMWORDSIZE)
      port map(
ADD => cache_addr,
ME => en,
WE => cache_rwn,
DI => cache_datain_part`'CHIPNR,
DO => cache_dataout_part`'CHIPNR);
})
   ram`'rest : ramcell
      generic map(
addr_range => eval(LOG(eval(IFECACHESIZE-1))-1),
words => IFECACHESIZE,
wordlength => REMRAMWORDSIZE)
      port map(
ADD => cache_addr,
```

```
ME => en,
WE => cache_rwn,
DI => cache_datain_partrest,
DO => cache_dataout_partrest);

define({UPPER}, INSTRSIZE)
define({LOWER}, eval(INSTRSIZE-MAXRAMWORDSIZE))

FOR(CHIPNR, 1, NUM_OF_MAXRAMS,
{
   cache_datain_part`'CHIPNR <= cache_datain(UPPER-1 downto LOWER);
   cache_dataout(UPPER-1 downto LOWER) <= cache_dataout_part`'CHIPNR;

define({UPPER}, LOWER)
define({LOWER}, eval(LOWER-MAXRAMWORDSIZE))
})

   cache_datain_partrest <= ifelse(eval(UPPER>0), 1, {cache_datain(UPPER-1 downto 0) & }) cache_tagin;

ifelse(eval(UPPER > 0),1,
{   cache_dataout(UPPER-1 downto 0) <= cache_dataout_partrest(REMRAMWORDSIZE-1 downto IFETAGSIZE); })

   cache_tagout <= cache_dataout_partrest(IFETAGSIZE-1 downto 0);

undefine({UPPER})
undefine({LOWER})

end;
-------------------------------------------------------------------------------
-- Ifetch FU FUNR
--
--`{ parameter dependencies: BUSWIDTH PARMOVES SINGLEGUARD CIFEFU}'FUNR...
-------------------------------------------------------------------------------
--
-- R1 : PC
-- R2 : PCC

library IEEE;
use IEEE.Std_Logic_1164.all;

entity ifefu`'FUNR is
   port(
FOR(BUSNR, 0, PARMOVES-1,
--
_ifelse(_eval(CIFEFU`'FUNR`'_TBUS`'BUSNR && (CIFEFU`'FUNR`'_R1BUS`'BUSNR ||
        CIFEFU`'FUNR`'_R2BUS`'BUSNR)), 1,
        databus`'BUSNR : inout std_logic_vector(eval(BUSWIDTH-1) downto 0);,
        _eval(CIFEFU`'FUNR`'_TBUS`'BUSNR && !(CIFEFU`'FUNR`'_R1BUS`'BUSNR ||
        CIFEFU`'FUNR`'_R2BUS`'BUSNR)), 1,
        databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);,
        _eval(!CIFEFU`'FUNR`'_TBUS`'BUSNR && (CIFEFU`'FUNR`'_R1BUS`'BUSNR ||
        CIFEFU`'FUNR`'_R2BUS`'BUSNR)), 1,
        databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);)
_ifelse(_eval(CIFEFU`'FUNR`'_TBUS`'BUSNR || CIFEFU`'FUNR`'_R2BUS`'BUSNR), 1,
        squash`'BUSNR : in std_logic;)
)
FOR(BUSNR, 0, PARMOVES-1,
{ }srcbus`'BUSNR : out std_logic_vector(_sub(SRCSIZE`'BUSNR)-1 downto 0);
{ }dstbus`'BUSNR : out std_logic_vector(_sub(DSTSIZE`'BUSNR)-1 downto 0);
ifdef({SINGLEGUARD},,`'guardspec`'BUSNR : out std_logic_vector(eval(GUARDSPECSIZE-1) downto 0);)
)
ifdef({SINGLEGUARD},`'guardspec : out std_logic_vector(eval(GUARDSPECSIZE-1) downto 0);)

squall : out std_logic;

iin : in std_logic_vector(eval(INSTRSIZE-1) downto 0);
ia : out std_logic_vector(eval(IFEADDRSIZE-1) downto 0);
ias : out std_logic;
il : out std_logic;

int : in std_logic;

glock : in std_logic;
glockreq : out std_logic;
exc : in std_logic;
ck : in std_logic;
rst : in std_logic);
end ifefu`'FUNR;

use work.sockets.all;
use work.internals.all;

architecture func of ifefu`'FUNR is
define({NRTBUSES}, translit(NIFEFU\FUNR\_T,\))
define({NRR1BUSES}, translit(NIFEFU\FUNR\_R1,\))
define({NRR2BUSES}, translit(NIFEFU\FUNR\_R2,\))
   for tsock:inskne`'NRTBUSES use entity work.inskne`'NRTBUSES`'(ARCHSTYLE);
```

```
   for r1sock:outreg`'NRR1BUSES use entity work.outreg`'NRR1BUSES`'(ARCHSTYLE);
   for r2sock:outsel`'NRR2BUSES use entity work.outsel`'NRR2BUSES`'(ARCHSTYLE);
   for internals:ifefui use entity work.ifefui(ARCHSTYLE);

   signal intern_tdata : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_topcode : std_logic_vector(LOG(eval(IFEFUTS-1))-1 downto 0);
   signal intern_tdataact : std_logic;
   signal intern_tsquash : std_logic_vector(NRTBUSES-1 downto 0);

   signal intern_r1data : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_r2data : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_r2squash : std_logic_vector(NRR2BUSES-1 downto 0);
   signal intern_r2outsels : std_logic;

translit(
FOR(BUSNR, 0, PARMOVES-1,
{   }signal ifefui_srcbus`'BUSNR : std_logic_vector(_sub(SRCSIZE`'BUSNR)-1 downto 0);
{   }signal ifefui_dstbus`'BUSNR : std_logic_vector(_sub(DSTSIZE`'BUSNR)-1 downto 0);
),\)

begin
   tsock:inskne`'NRTBUSES
define({BSUBST}, translit(BIFEFU\FUNR\_T,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NRTBUSES-1,
{ }ldstsize`'BUSNR => _sub(DSTSIZE`'BSUBST\BUSNR){,}
{ }lowid`'BUSNR => _sub(AIFEFU\FUNR\_TBUS`'BSUBST\BUSNR){,}
){ }logopcodes => LOG(eval(IFEFUTS-1))) ,\)
translit(
{      }port map(
FOR(BUSNR, 0, NRTBUSES-1,
{ }databus`'BUSNR => databus`'BSUBST\BUSNR{,}
{ }dstbus`'BUSNR => ifefui_dstbus`'BSUBST\BUSNR{,}
){ }squash => intern_tsquash{,}
{ }glock => glock{,}
{ }ck => ck,
{ }data => intern_tdata,
{ }opcode => intern_topcode,
{ }dataact => intern_tdataact );
,\)
translit(
FOR(BUSNR, 0, NRTBUSES-1, {   }intern_tsquash(BUSNR) <= squash`'BSUBST\BUSNR;
),\)

   r1sock:outreg`'NRR1BUSES
define({BSUBST}, translit(BIFEFU\FUNR\_R1,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NRR1BUSES-2,
{ }lsrcsize`'BUSNR => _sub(SRCSIZE`'BSUBST\BUSNR){,}
{ }lowid`'BUSNR => _sub(AIFEFU\FUNR\_R1BUS`'BSUBST\BUSNR){,}
){ }lsrcsize`'eval(NRR1BUSES-1) =>_sub(SRCSIZE`'BSUBST\eval(NRR1BUSES-1)){,}
{ }lowid`'eval(NRR1BUSES-1) => _sub(AIFEFU\FUNR\_R1BUS`'BSUBST\eval(NRR1BUSES-1))),\)
translit(
{      }port map(
FOR(BUSNR, 0, NRR1BUSES-1,
{ }databus`'BUSNR => databus`'BSUBST\BUSNR{,}
{ }srcbus`'BUSNR => ifefui_srcbus`'BSUBST\BUSNR{,}
){ }glock => glock{,}
{ }ck => ck{,}
{ }data => intern_r1data);
,\)

   r2sock:outsel`'NRR2BUSES
define({BSUBST}, translit(BIFEFU\FUNR\_R2,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NRR2BUSES-2,
{ }lsrcsize`'BUSNR => _sub(SRCSIZE`'BSUBST\BUSNR){,}
{ }lowid`'BUSNR => _sub(AIFEFU\FUNR\_R2BUS`'BSUBST\BUSNR){,}
){ }lsrcsize`'eval(NRR2BUSES-1) =>_sub(SRCSIZE`'BSUBST\eval(NRR2BUSES-1)){,}
{ }lowid`'eval(NRR2BUSES-1) => _sub(AIFEFU\FUNR\_R2BUS`'BSUBST\eval(NRR2BUSES-1))),\)
translit(
{      }port map(
FOR(BUSNR, 0, NRR2BUSES-1,
{ }databus`'BUSNR => databus`'BSUBST\BUSNR{,}
{ }srcbus`'BUSNR => ifefui_srcbus`'BSUBST\BUSNR{,}
){ }squash => intern_r2squash{,}
{ }glock => glock{,}
{ }ck => ck{,}
{ }data => intern_r2data{,}
{ }outsels => intern_r2outsels);
,\)
translit(
FOR(BUSNR, 0, NRR2BUSES-1, {   }intern_r2squash(BUSNR) <= squash`'BSUBST\BUSNR;
),\)
```

```
   internals:ifefui
      port map(
{ }tdata => intern_tdata,
{ }topcode => intern_topcode,
{ }tdataact => intern_tdataact,

{ }r1data => intern_r1data,
{ }r2data => intern_r2data,
{ }r2outsels => intern_r2outsels,
FOR(BUSNR, 0, PARMOVES-1,
{ }srcbus`'BUSNR => ifefui_srcbus`'BUSNR{,}
{ }dstbus`'BUSNR => ifefui_dstbus`'BUSNR{,}
ifdef({SINGLEGUARD},,{ guardspec`'BUSNR => guardspec`'BUSNR{,}})
)ifdef({SINGLEGUARD},{ guardspec => guardspec{,}})
{ }squall => squall,
{ }iin => iin,
{ }ia => ia,
{ }ias => ias,
{ }il => il,
{ }int => int,
--#{ }cch_a => cch_a,
--#{ }cch_dt => cch_dt,
--#{ }cch_qi => cch_qi,
--#{ }cch_qt => cch_qt,
--#{ }cch_rwn => cch_rwn,
{ }glock => glock,
{ }glockreq => glockreq,
{ }exc => exc,
{ }ck => ck,
{ }rst => rst );

FOR(BUSNR, 0, PARMOVES-1,
{   }srcbus`'BUSNR <= ifefui_srcbus`'BUSNR;
{   }dstbus`'BUSNR <= ifefui_dstbus`'BUSNR;
)
end func;
------------------------------------------------------------------------------
-- Ifetch FU internals
--
--`{ parameter dependencies: BUSWIDTH IFEADDRSIZE SINGLEGUARD GUARDSPECSIZE
--  IFECACHESIZE}'
------------------------------------------------------------------------------

library IEEE;
use IEEE.Std_Logic_1164.all;
use WORK.ifetchcontrol.all;

entity ifefui is
   port(
tdata : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
topcode : in std_logic_vector(LOG(eval(IFEFUTS-1))-1 downto 0);
tdataact : in std_logic;

r1data : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
r2data : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
r2outsels : in std_logic;
translit(
FOR(BUSNR, 0, PARMOVES-1,
{ }srcbus`'BUSNR : out std_logic_vector(SRCSIZE\BUSNR-1 downto 0);
{ }dstbus`'BUSNR : out std_logic_vector(DSTSIZE\BUSNR-1 downto 0);
ifdef({SINGLEGUARD},,
{ }guardspec`'BUSNR : out std_logic_vector(eval(GUARDSPECSIZE-1) downto 0);)
),\)
ifdef({SINGLEGUARD},
{ }guardspec : out std_logic_vector(eval(GUARDSPECSIZE-1) downto 0);)
{ }squall : out std_logic;
{ }iin : in std_logic_vector(eval(INSTRSIZE-1) downto 0); -- instr. data inputs
{ }ia : out std_logic_vector(eval(IFEADDRSIZE-1) downto 0); -- instr. address
{ }ias : out std_logic; -- instr. addr. strobe
{ }il : out std_logic; -- instr. latch

{ }int : in std_logic; -- external interrupt

--#{ }cch_a : out std_logic_vector(LOG(eval(IFECACHESIZE-1))-1 downto 0); -- cache address
--#{ }cch_dt : out std_logic_vector(eval(IFETAGSIZE-1) downto 0); -- cache tag data inp.
--#
--#{ }cch_qi : in std_logic_vector(eval(INSTRSIZE-1) downto 0); -- cache ins. data outp.
--#{ }cch_qt : in std_logic_vector(eval(IFETAGSIZE-1) downto 0); -- cache tag data outp.
--#
--#{ }cch_rwn : out std_logic; -- cache read/write

{ }glock : in std_logic;  -- unconditional lock
{ }glockreq : out std_logic;  -- glob. lock request
{ }exc : in std_logic; -- exc. signal from netcontrol
{ }ck : in std_logic;
{ }rst : in std_logic);
```

```
end ifefui;


architecture func of ifefui is

-- pc registers for pipeline and pc chain
   signal pcifreg, pcdcreg, pccifreg, pccdcreg,
  pccmvreg : std_logic_vector(eval(IFEADDRSIZE-1) downto 0);

-- instruction register and its input
   signal insreg, cchpth_insreg : std_logic_vector(eval(INSTRSIZE-1) downto 0);

-- ctlreg (cache enable)
   signal ctlreg : std_logic;

-- pc register enable signals
   signal pcccctl_pccif_en, pccctl_pccdc_en, pccctl_pccmv_en,
pccctl_excmode : std_logic;
   signal regctl_ctlreg_en, regctl_pcen : std_logic;

-- pc registers from pc selector
   signal pcsel_newpcc, pcsel_newpc : std_logic_vector(eval(IFEADDRSIZE-1) downto 0);

-- pc incrementer output
   signal pcincr_y : std_logic_vector(eval(IFEADDRSIZE-1) downto 0);

-- cache control signals
   signal cchctl_cwrite, cchctl_insext, cchctl_insen,
cchctl_insenforce : std_logic;
   signal cchpth_insvalid : std_logic;

-- signals to/from instruction cache
   signal cch_a : std_logic_vector(LOG(eval(IFECACHESIZE-1))-1 downto 0); -- cache address
   signal cch_dt : std_logic_vector(eval(IFETAGSIZE-1) downto 0); -- cache tag data inp.
   signal cch_qi : std_logic_vector(eval(INSTRSIZE-1) downto 0); -- cache ins. data outp.
   signal cch_qt : std_logic_vector(eval(IFETAGSIZE-1) downto 0); -- cache tag data outp.
   signal cch_rwn : std_logic; -- cache read/write

   for regcontrol:regctl use entity work.regctl(ARCHSTYLE);
   for pcincrement:pcincr use entity work.pcincr(ARCHSTYLE);
   for pcselect:pcsel use entity work.pcsel(ARCHSTYLE);
   for cachepath:cchpth use entity work.cchpth(ARCHSTYLE);
   for cachecontrol:cchctl use entity work.cchctl(ARCHSTYLE);
   for pcccontrol:pccctl use entity work.pccctl(ARCHSTYLE);

begin

-- register loads
   process
   begin
      wait until ck'event and ck='1';

      if (cchctl_insen='1') then
 insreg <= cchpth_insreg;
      end if;
      if (pccctl_pccmv_en='1') then
 pccmvreg <= pccdcreg;
      end if;
      if (pccctl_pccdc_en='1') then
 pccdcreg <= pccifreg;
      end if;
      if (pccctl_pccif_en='1') then
 pccifreg <= pcsel_newpcc;
      end if;
      if (regctl_ctlreg_en='1') then
 ctlreg <= tdata(0);
      end if;
      if not(rst='1') then
 ctlreg <= '0';
      end if;
      if (regctl_pcen='1') then
 pcdcreg <= pcifreg;
 pcifreg <= pcsel_newpc;
      end if;
   end process;

-- output sockets

ifdef({IFESHIFTADDR},{
define({TEMP1}, eval(LOG((eval(INSTRSIZE-1))/BUSWIDTH)) )
ifelse(eval(TEMP1>0), 1,{
   r1data(eval(TEMP1-1) downto 0) <= "FOR(CNT,0,TEMP1-1,`0')";
   r2data(eval(TEMP1-1) downto 0) <= "FOR(CNT,0,TEMP1-1,`0')";})

define({TEMP2}, eval(IFEADDRSIZE-1+TEMP1));
   r1data(TEMP2 downto TEMP1) <= pcifreg;
   r2data(TEMP2 downto TEMP1) <= pccmvreg;
```

```
ifelse(eval(TEMP2<BUSWIDTH-1), 1, {
   r1data(eval(BUSWIDTH-1) downto eval(TEMP2+1)) <= "FOR(CNT,TEMP2+1,BUSWIDTH-1,`0')";
   r2data(eval(BUSWIDTH-1) downto eval(TEMP2+1)) <= "FOR(CNT,TEMP2+1,BUSWIDTH-1,`0')";})
},{
   r1data(eval(IFEADDRSIZE-1) downto 0) <= pcifreg;
   r2data(eval(IFEADDRSIZE-1) downto 0) <= pccmvreg;

ifelse(eval(IFEADDRSIZE-1<BUSWIDTH-1), 1, {
   r1data(eval(BUSWIDTH-1) downto IFEADDRSIZE) <= "FOR(CNT,IFEADDRSIZE,BUSWIDTH-1,`0')";
   r2data(eval(BUSWIDTH-1) downto IFEADDRSIZE) <= "FOR(CNT,IFEADDRSIZE,BUSWIDTH-1,`0')";})
})

-- register control

   regcontrol:regctl
    port map(
glock => glock,
rst => rst,
tdataact => tdataact,
nopcwrite => topcode(2),
noctlsel => topcode(1),
pcen => regctl_pcen);

-- PC incrementer

   pcincrement:pcincr
    port map( i => pcifreg,
      y => pcincr_y);

-- PC selector

   pcselect:pcsel
    port map(
incpc => pcincr_y,
tdata => ifdef({IFESHIFTADDR},{tdata(TEMP2 downto TEMP1)},{tdata(eval(IFEADDRSIZE-1) downto 0)}),
tdataact => tdataact,
offonly => topcode(0),
nopcwrite => topcode(2),
rst => rst,
excmode => pccctl_excmode,
exc => exc,
trap => topcode(1),
int => int,
newpc => pcsel_newpc,
newpcc => pcsel_newpcc);

-- cache RAM, path and control

   cacheram:icacheram
    port map(
cache_addr => cch_a,
cache_datain => iin,
cache_tagin => cch_dt,
cache_dataout => cch_qi,
cache_tagout => cch_qt,
cache_rwn => cch_rwn ifdef({IFECACHERAMASYN},,{,
ck => ck})
);

   cachepath:cchpth
    port map(
ifdef({IFECACHERAMASYN},,
{ newifca => pcsel_newpc(LOG(eval(IFECACHESIZE-1))-1 downto 0),
}) ifa => pcifreg,
dca => pcdcreg,
cwrite => cchctl_cwrite,
rama => cch_a,
dt => cch_dt,
qt => cch_qt,
cenabled => ctlreg,
insenforce => cchctl_insenforce,
insen => cchctl_insen,
insvalid => cchpth_insvalid,
iin => iin,
inscache => cch_qi,
insext => cchctl_insext,
ifdef({IFECACHERAMASYN},,
{ glock => glock,
}) rst => rst,
ck => ck);

   cachecontrol:cchctl
    port map(
insvalid => cchpth_insvalid,
rwn => cch_rwn,
cwrite => cchctl_cwrite,
```

```
ias => ias,
il => il,
insext => cchctl_insext,
insen => cchctl_insen,
insenforce => cchctl_insenforce,
glockreq => glockreq,
glock => glock,
rst => rst,
ck => ck);

   ia <= pcdcreg;

   pcccontrol:pccctl
    port map(
pccif_en => pccctl_pccif_en,
pccdc_en => pccctl_pccdc_en,
pccmv_en => pccctl_pccmv_en,
excmode => pccctl_excmode,
squall => squall,
tdataact => tdataact,
nopcwrite => topcode(2),
trap => topcode(1),
r2outsels => r2outsels,
glock => glock,
exc => exc,
int => int,
rst => rst,
ck => ck);

_define({LOWER}, 0)
_define({UPPER}, 0)

ifdef({SINGLEGUARD},
{FOR(BUSNR,0,PARMOVES-1,
_define(‘UPPER’, _eval(UPPER+‘DSTSIZE’BUSNR))
   ‘dstbus’BUSNR <= insreg(_eval(UPPER-1) downto LOWER);
_define(‘LOWER’, UPPER)
_define(‘UPPER’, _eval(UPPER+‘SRCSIZE’BUSNR))
   ‘srcbus’BUSNR <= insreg(_eval(UPPER-1) downto LOWER);
_define(‘LOWER’, UPPER)
)
 _define(‘UPPER’, _eval(UPPER+GUARDSPECSIZE))
   guardspec <= insreg(_eval(UPPER-1) downto LOWER);
},

{FOR(BUSNR,0,PARMOVES-1,
_define(‘UPPER’, _eval(UPPER+DSTSIZE‘’BUSNR))
   dstbus‘’BUSNR <= insreg(_eval(UPPER-1) downto LOWER);
_define(‘LOWER’, UPPER)
_define(‘UPPER’, _eval(UPPER+SRCSIZE‘’BUSNR))
   srcbus‘’BUSNR <= insreg(_eval(UPPER-1) downto LOWER);
_define(‘LOWER’, UPPER)
_define(‘UPPER’, _eval(UPPER+GUARDSPECSIZE))
   guardspec‘’BUSNR <= insreg(_eval(UPPER-1) downto LOWER);
_define(‘LOWER’, UPPER)
)
})

end func;
------------------------------------------------------------------------------
-- Immediate FU FUNR
--
--‘{ parameter dependencies: BUSWIDTH PARMOVES CIMMFU}’FUNR...
------------------------------------------------------------------------------

library IEEE;
use IEEE.Std_Logic_1164.all;

entity immfu‘’FUNR is
   port(
FOR(BUSNR, 0, PARMOVES-1,
--
_ifelse(CIMMFU‘’FUNR‘’_RBUS‘’BUSNR, 1,
databus‘’BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
srcbus‘’BUSNR : in std_logic_vector(SRCSIZE‘’BUSNR-1 downto 0);)
)--
glock : in std_logic;
ck : in std_logic);

end immfu‘’FUNR;

use work.sockets.all;
use work.internals.all;

architecture func of immfu‘’FUNR is
define({NRRBUSES}, translit(NIMMFU\FUNR\_R,\))
```

```
   for rsock:outimm`'NRRBUSES use entity WORK.outimm`'NRRBUSES`'(ARCHSTYLE);
   for fu_internals:immfui`'NRRBUSES use entity WORK.immfui`'NRRBUSES`'(ARCHSTYLE);
FOR(BUSNR, 0, NRRBUSES-1,
{   }signal rsock_data`'BUSNR : std_logic_vector(eval(BUSWIDTH-1) downto 0);
{   }signal rsock_osrcbus`'BUSNR : std_logic_vector(LOG(eval(IMMEDIATES-1))-1 downto 0);
)
begin
   rsock:outimm`'NRRBUSES
define({BSUBST}, translit(BIMMFU\FUNR\_R,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NRRBUSES-2,
{ }lsrcsize`'BUSNR => _sub(SRCSIZE`'BSUBST\BUSNR){,}
{ }lowid`'BUSNR => _sub(AIMMFU\FUNR\_RBUS`'BSUBST\BUSNR){,}
){ }lsrcsize`'eval(NRRBUSES-1) => _sub(SRCSIZE`'BSUBST\eval(NRRBUSES-1)){,}
{ }lowid`'eval(NRRBUSES-1) => _sub(AIMMFU\FUNR\_RBUS`'BSUBST\eval(NRRBUSES-1))),\)
translit(
{      }port map(
FOR(BUSNR, 0, NRRBUSES-1,
{ }databus`'BUSNR => databus`'BSUBST\BUSNR{,}
{ }srcbus`'BUSNR => srcbus`'BSUBST\BUSNR{,}
)FOR(BUSNR, 0, NRRBUSES-1,
{ }data`'BUSNR => rsock_data`'BSUBST\BUSNR{,}
{ }osrcbus`'BUSNR => rsock_osrcbus`'BSUBST\BUSNR{,}
){ }glock => glock,
{ }ck => ck);,\)

   fu_internals:immfui`'NRRBUSES
      port map(
FOR(BUSNR, 0, NRRBUSES-1,
{ }rdata`'BUSNR => rsock_data`'BUSNR{,}
{ }srcbus`'BUSNR => rsock_osrcbus`'BUSNR{,}
) glock => glock,
ck => ck);

end func;
FOR(COUNTER,1,PARMOVES,
{define({FUINR},} COUNTER {)
-------------------------------------------------------------------------------
-- Immediate FU Internals for Immfu FUINR
--
--`{ parameter dependencies: BUSWIDTH IMMEDIATES}'
--
-- Note: these internals are different for each FU
-------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity immfui`'FUINR is
   port(
FOR(NDX, 0, FUINR-1,{ }rdata`'NDX : out std_logic_vector(BUSWIDTH-1 downto 0);
{ }srcbus`'NDX : in std_logic_vector(LOG(eval(IMMEDIATES-1))-1 downto 0);
) glock : in std_logic;
ck : in std_logic);
end immfui`'FUINR;


architecture func of immfui`'FUINR is
FOR(NDX, 0, FUINR-1,{
   }signal immreg`'NDX : std_logic_vector(LOG(eval(IMMEDIATES-1))-1 downto 0);)

begin

   process
   begin
      wait until ck'event and ck='1';
      if not (glock = '1') then
FOR(NDX, 0, FUINR-1,{  }immreg`'NDX <= srcbus`'NDX;
)      end if;
   end process;
FOR(NDX, 0, FUINR-1,{
   }rdata`'NDX`'(LOG(eval(IMMEDIATES-1))-1 downto 0) <= immreg`'NDX;
   rdata`'NDX`'(BUSWIDTH-1 downto LOG(eval(IMMEDIATES-1))) <= "FOR(CNT,LOG(eval(IMMEDIATES-1)),BUSWIDTH-1,0)";)
end func;
})
FOR(COUNTER, 1, PARMOVES,
{define({SOCKNR},} COUNTER{)
-------------------------------------------------------------------------------
-- input socket without BSY (locking support) for SOCKNR buses
--
--`{ parameter dependencies: BUSWIDTH generic parameters}'
-------------------------------------------------------------------------------

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

```
entity insknb`'SOCKNR is
   generic(
FOR(BUSNR, 0, SOCKNR-1,
{ }ldstsize`'BUSNR : integer;
{ }lowid`'BUSNR : integer;
){ }logopcodes : integer);

   port(
FOR(BUSNR, 0, SOCKNR-1,
{ }databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
{ }dstbus`'BUSNR : in std_logic_vector(ldstsize`'BUSNR-1 downto 0);
) excreq : out std_logic;
squash : in std_logic_vector(eval(SOCKNR-1) downto 0);
glock : in std_logic;
ck : in std_logic;

data : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
opcode : out std_logic_vector(logopcodes-1 downto 0);
fucls : in std_logic;
outsels : in std_logic;
dataact : out std_logic);
end insknb`'SOCKNR;


architecture func of insknb`'SOCKNR is
   signal selectl : std_logic_vector(eval(SOCKNR-1) downto 0);
FOR(BUSNR, 0, SOCKNR-1,
{   }signal opcodel`'BUSNR : std_logic_vector(logopcodes-1 downto 0);
)   signal insels : std_logic;
begin
   insels <= (not squash(0) and selectl(0))
FOR(BUSNR, 1, SOCKNR-1,
{   }or (not squash(BUSNR) and selectl(BUSNR))
);
   dataact <= not glock and insels;
   excreq <= fucls and (insels and not outsels);

   data <= FOR(BUSNR, 0, SOCKNR-1,{
}databus`'BUSNR when ((not squash(BUSNR) and selectl(BUSNR)) = '1') else )
(others => '-');
   opcode <= FOR(BUSNR, 0, SOCKNR-1,{
}opcodel`'BUSNR when ((not squash(BUSNR) and selectl(BUSNR)) = '1') else )
(others => '-');

   process
FOR(BUSNR, 0, SOCKNR-1,
{      }variable id`'BUSNR : std_logic_vector(ldstsize`'BUSNR-1 downto 0);
)   begin
      wait until ck'event and ck='1';
      if (glock = '0') then
FOR(BUSNR, 0, SOCKNR-1,
{  }opcodel`'BUSNR <= dstbus`'BUSNR`'(logopcodes-1 downto 0);
{  }id`'BUSNR := std_logic_vector(conv_unsigned(lowid`'BUSNR, id`'BUSNR'length));
{  }if (dstbus`'BUSNR`'(ldstsize`'BUSNR-1 downto logopcodes) =
{ }id`'BUSNR`'(ldstsize`'BUSNR-1 downto logopcodes)) then
{     }selectl(BUSNR) <= '1';
{  }else
{     }selectl(BUSNR) <= '0';
{  }end if;
)      end if;
   end process;
end func;
})
FOR(COUNTER, 1, PARMOVES,
{define({SOCKNR},} COUNTER()
-----------------------------------------------------------------------------
-- input sockets without BSY (locking support) and exceptions,
-- dataact without glock, for SOCKNT buses
--
--`{ parameter dependencies: BUSWIDTH generic parameters}'
-----------------------------------------------------------------------------

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

entity inskne`'SOCKNR is
   generic(
FOR(BUSNR, 0, SOCKNR-1,
{ }ldstsize`'BUSNR : integer;
{ }lowid`'BUSNR : integer;
){ }logopcodes : integer);

   port(
FOR(BUSNR, 0, SOCKNR-1,
{ }databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
```

```
{ }dstbus`'BUSNR : in std_logic_vector(ldstsize`'BUSNR-1 downto 0);
) squash : in std_logic_vector(eval(SOCKNR-1) downto 0);
glock : in std_logic;
ck : in std_logic;

data : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
opcode : out std_logic_vector(logopcodes-1 downto 0);
dataact : out std_logic);
end inskne`'SOCKNR;


architecture func of inskne`'SOCKNR is
   signal selectl : std_logic_vector(eval(SOCKNR-1) downto 0);
FOR(BUSNR, 0, SOCKNR-1,
{   }signal opcodel`'BUSNR : std_logic_vector(logopcodes-1 downto 0);
)   signal insels : std_logic;
begin
   insels <= (not squash(0) and selectl(0))
FOR(BUSNR, 1, SOCKNR-1,
{   }or (not squash(BUSNR) and selectl(BUSNR))
);
   dataact <= insels;

   data <= FOR(BUSNR, 0, SOCKNR-1,{
}databus`'BUSNR when ((not squash(BUSNR) and selectl(BUSNR)) = '1') else )
(others => '-');
   opcode <= FOR(BUSNR, 0, SOCKNR-1,{
}opcodel`'BUSNR when ((not squash(BUSNR) and selectl(BUSNR)) = '1') else )
(others => '-');

   process
FOR(BUSNR, 0, SOCKNR-1,
{      }variable id`'BUSNR : std_logic_vector(ldstsize`'BUSNR-1 downto 0);
)   begin
      wait until ck'event and ck='1';
      if (glock = '0') then
FOR(BUSNR, 0, SOCKNR-1,
{  }opcodel`'BUSNR <= dstbus`'BUSNR`'(logopcodes-1 downto 0);
{  }id`'BUSNR := std_logic_vector(conv_unsigned(lowid`'BUSNR, id`'BUSNR'length));
{  }if (dstbus`'BUSNR`'(ldstsize`'BUSNR-1 downto logopcodes) =
{ }id`'BUSNR`'(ldstsize`'BUSNR-1 downto logopcodes)) then
{     }selectl(BUSNR) <= '1';
{  }else
{     }selectl(BUSNR) <= '0';
{  }end if;
)      end if;
   end process;
end func;
})
FOR(COUNTER, 1, PARMOVES,
{define({SOCKNR},} COUNTER{)
------------------------------------------------------------------------------
-- input sockets without opcode register for SOCKNR buses
--
--`{ parameter dependencies: BUSWIDTH generic parameters}'
------------------------------------------------------------------------------

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

entity insnop`'SOCKNR is
   generic(
FOR(BUSNR, 0, SOCKNR-2,
{ }ldstsize`'BUSNR : integer := 7;
{ }lowid`'BUSNR : integer := 83;
){ }ldstsize`'eval(SOCKNR-1) : integer := 7;
{ }lowid`'eval(SOCKNR-1) : integer := 85);

   port(
FOR(BUSNR, 0, SOCKNR-1,
{ }databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
{ }dstbus`'BUSNR : in std_logic_vector(ldstsize`'BUSNR-1 downto 0);
) excreq : out std_logic;
llockreq : out std_logic_vector(eval(SOCKNR-1) downto 0);
squash : in std_logic_vector(eval(SOCKNR-1) downto 0);
glock : in std_logic;
ck : in std_logic;

data : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
fucls : in std_logic;
fubsy : in std_logic;
outsels : in std_logic;
dataact : out std_logic);
end insnop`'SOCKNR;

architecture func of insnop`'SOCKNR is
```

```
   signal selectl : std_logic_vector(eval(SOCKNR-1) downto 0);
   signal insels : std_logic;
begin
   insels <= (not squash(0) and selectl(0))
FOR(BUSNR, 1, SOCKNR-1,
{   }or (not squash(BUSNR) and selectl(BUSNR))
);
   dataact <= not glock and insels;
FOR(BUSNR, 0, SOCKNR-1,
{   }llockreq(BUSNR) <= selectl(BUSNR) and fubsy;
)   excreq <= fucls and (insels and not outsels);

   data <= FOR(BUSNR, 0, SOCKNR-1,{
}databus`'BUSNR when ((not squash(BUSNR) and selectl(BUSNR)) = '1') else )
(others => '-');

   process
FOR(BUSNR, 0, SOCKNR-1,
{      }variable id`'BUSNR : std_logic_vector(ldstsize`'BUSNR-1 downto 0);
)   begin
      wait until ck'event and ck='1';
      if (glock = '0') then
FOR(BUSNR, 0, SOCKNR-1,
{  }id`'BUSNR := std_logic_vector(conv_unsigned(lowid`'BUSNR, id`'BUSNR'length));
{  }if (dstbus`'BUSNR=id`'BUSNR) then
{     }selectl(BUSNR) <= '1';
{  }else
{     }selectl(BUSNR) <= '0';
{  }end if;
)      end if;
   end process;
end func;
})
FOR(COUNTER, 1, PARMOVES,
{define({SOCKNR},} COUNTER{)
------------------------------------------------------------------------------
-- input socket for SOCKNR buses
--
--`{ parameter dependencies: BUSWIDTH generic parameters}'
------------------------------------------------------------------------------

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

entity insock`'SOCKNR is
   generic(
FOR(BUSNR, 0, SOCKNR-1,
{ }ldstsize`'BUSNR : integer;
{ }lowid`'BUSNR : integer;
){ }logopcodes : integer);

   port(
FOR(BUSNR, 0, SOCKNR-1,
{ }databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
{ }dstbus`'BUSNR : in std_logic_vector(ldstsize`'BUSNR-1 downto 0);
) excreq : out std_logic;
llockreq : out std_logic_vector(eval(SOCKNR-1) downto 0);
squash : in std_logic_vector(eval(SOCKNR-1) downto 0);
glock : in std_logic;
ck : in std_logic;

data : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
opcode : out std_logic_vector(logopcodes-1 downto 0);
fucls : in std_logic;
fubsy : in std_logic;
outsels : in std_logic;
dataact : out std_logic);
end insock`'SOCKNR;


architecture func of insock`'SOCKNR is
   signal selectl : std_logic_vector(eval(SOCKNR-1) downto 0);
FOR(BUSNR, 0, SOCKNR-1,
{   }signal opcodel`'BUSNR : std_logic_vector(logopcodes-1 downto 0);
)   signal insels : std_logic;
begin
   insels <= (not squash(0) and selectl(0))
FOR(BUSNR, 1, SOCKNR-1,
{   }or (not squash(BUSNR) and selectl(BUSNR))
);
   dataact <= not glock and insels;
FOR(BUSNR, 0, SOCKNR-1,
{   }llockreq(BUSNR) <= selectl(BUSNR) and fubsy;
)   excreq <= fucls and (insels and not outsels);

   data <= FOR(BUSNR, 0, SOCKNR-1,{
```

```
}databus`'BUSNR when ((not squash(BUSNR) and selectl(BUSNR)) = '1') else )
(others => '-');
   opcode <= FOR(BUSNR, 0, SOCKNR-1,{
}opcodel`'BUSNR when ((not squash(BUSNR) and selectl(BUSNR)) = '1') else )
(others => '-');


   process
FOR(BUSNR, 0, SOCKNR-1,
{      }variable id`'BUSNR : std_logic_vector(ldstsize`'BUSNR-1 downto 0);
)   begin
      wait until ck'event and ck='1';
      if (glock = '0') then
FOR(BUSNR, 0, SOCKNR-1,
{  }opcodel`'BUSNR <= dstbus`'BUSNR`'(logopcodes-1 downto 0);
{  }id`'BUSNR := std_logic_vector(conv_unsigned(lowid`'BUSNR, id`'BUSNR'length));
{  }if (dstbus`'BUSNR`'(ldstsize`'BUSNR-1 downto logopcodes) =
{ }id`'BUSNR`'(ldstsize`'BUSNR-1 downto logopcodes)) then
{     }selectl(BUSNR) <= '1';
{  }else
{     }selectl(BUSNR) <= '0';
{  }end if;
)      end if;
   end process;
end func;
})
FOR(COUNTER, 1, PARMOVES,
{define({SOCKNR},} COUNTER{)
-----------------------------------------------------------------------------
-- input sockets without opcode register and BSY, for SOCKNR buses
--
--`{ parameter dependencies: BUSWIDTH generic parameters}'
-----------------------------------------------------------------------------

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

entity inspnb`'SOCKNR is
   generic(
FOR(BUSNR, 0, SOCKNR-2,
{ }ldstsize`'BUSNR : integer := 7;
{ }lowid`'BUSNR : integer := 83;
){ }ldstsize`'eval(SOCKNR-1) : integer := 7;
{ }lowid`'eval(SOCKNR-1) : integer := 85);

   port(
FOR(BUSNR, 0, SOCKNR-1,
{ }databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
{ }dstbus`'BUSNR : in std_logic_vector(ldstsize`'BUSNR-1 downto 0);
) excreq : out std_logic;
squash : in std_logic_vector(eval(SOCKNR-1) downto 0);
glock : in std_logic;
ck : in std_logic;

data : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
fucls : in std_logic;
outsels : in std_logic;
dataact : out std_logic);
end inspnb`'SOCKNR;

architecture func of inspnb`'SOCKNR is
   signal selectl : std_logic_vector(eval(SOCKNR-1) downto 0);
   signal insels : std_logic;
begin
   insels <= (not squash(0) and selectl(0))
FOR(BUSNR, 1, SOCKNR-1,
{   }or (not squash(BUSNR) and selectl(BUSNR))
);
   dataact <= not glock and insels;
   excreq <= fucls and (insels and not outsels);

   data <= FOR(BUSNR, 0, SOCKNR-1,{
}databus`'BUSNR when ((not squash(BUSNR) and selectl(BUSNR)) = '1') else )
(others => '-');

   process
FOR(BUSNR, 0, SOCKNR-1,
{      }variable id`'BUSNR : std_logic_vector(ldstsize`'BUSNR-1 downto 0);
)   begin
      wait until ck'event and ck='1';
      if (glock = '0') then
FOR(BUSNR, 0, SOCKNR-1,
{  }id`'BUSNR := std_logic_vector(conv_unsigned(lowid`'BUSNR, id`'BUSNR'length));
{  }if (dstbus`'BUSNR=id`'BUSNR) then
{     }selectl(BUSNR) <= '1';
{  }else
```

```
{     }selectl(BUSNR) <= '0';
{  }end if;
)      end if;
   end process;
end func;
})
FOR(COUNTER, 1, PARMOVES,
{define({SOCKNR},} COUNTER{)
------------------------------------------------------------------------------
-- input sockets for registers, for SOCKNR buses
-- without opcode register, BSY (locking support) and exceptions.
--
--`{ parameter dependencies: BUSWIDTH generic parameters}'
------------------------------------------------------------------------------

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

entity insreg`'SOCKNR is
   generic(
FOR(BUSNR, 0, SOCKNR-2,
{ }ldstsize`'BUSNR : integer;
{ }lowid`'BUSNR : integer;
){ }ldstsize`'eval(SOCKNR-1) : integer;
{ }lowid`'eval(SOCKNR-1) : integer);

   port(
FOR(BUSNR, 0, SOCKNR-1,
{ }databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
{ }dstbus`'BUSNR : in std_logic_vector(ldstsize`'BUSNR-1 downto 0);
) squash : in std_logic_vector(eval(SOCKNR-1) downto 0);
glock : in std_logic;
ck : in std_logic;

data : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
dataact : out std_logic);
end insreg`'SOCKNR;

architecture func of insreg`'SOCKNR is
   signal selectl : std_logic_vector(eval(SOCKNR-1) downto 0);
   signal insels : std_logic;
begin
   insels <= (not squash(0) and selectl(0))
FOR(BUSNR, 1, SOCKNR-1,
{   }or (not squash(BUSNR) and selectl(BUSNR))
);
   dataact <= not glock and insels;

   data <= FOR(BUSNR, 0, SOCKNR-1,{
}databus`'BUSNR when ((not squash(BUSNR) and selectl(BUSNR)) = '1') else )
(others => '-');

   process
FOR(BUSNR, 0, SOCKNR-1,
{      }variable id`'BUSNR : std_logic_vector(ldstsize`'BUSNR-1 downto 0);
)   begin
      wait until ck'event and ck='1';
      if (glock = '0') then
FOR(BUSNR, 0, SOCKNR-1,
{  }id`'BUSNR := std_logic_vector(conv_unsigned(lowid`'BUSNR, id`'BUSNR'length));
{  }if (dstbus`'BUSNR=id`'BUSNR) then
{     }selectl(BUSNR) <= '1';
{  }else
{     }selectl(BUSNR) <= '0';
{  }end if;
)      end if;
   end process;
end func;
})
------------------------------------------------------------------------------
-- Integer FU FUNR
--
--`{ parameter dependencies: BUSWIDTH PARMOVES CINTFU}'FUNR...
------------------------------------------------------------------------------

library IEEE;
use IEEE.Std_Logic_1164.all;

entity intfu`'FUNR is
   port(
FOR(BUSNR, 0, PARMOVES-1,
--
_ifelse(_eval((CINTFU`'FUNR`'_TBUS`'BUSNR || CINTFU`'FUNR`'_OBUS`'BUSNR) &&
CINTFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : inout std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval((CINTFU`'FUNR`'_TBUS`'BUSNR || CINTFU`'FUNR`'_OBUS`'BUSNR) &&
```

```
!CINTFU‘’FUNR‘’_RBUS‘’BUSNR), 1,
databus‘’BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval(!(CINTFU‘’FUNR‘’_TBUS‘’BUSNR || CINTFU‘’FUNR‘’_OBUS‘’BUSNR) &&
CINTFU‘’FUNR‘’_RBUS‘’BUSNR), 1,
databus‘’BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);)
_ifelse(_eval(CINTFU‘’FUNR‘’_TBUS‘’BUSNR || CINTFU‘’FUNR‘’_OBUS‘’BUSNR ||
CINTFU‘’FUNR‘’_RBUS‘’BUSNR), 1,
squash‘’BUSNR : in std_logic;)
_ifelse(_eval(CINTFU‘’FUNR‘’_TBUS‘’BUSNR || CINTFU‘’FUNR‘’_OBUS‘’BUSNR), 1,
dstbus‘’BUSNR : in std_logic_vector(DSTSIZE‘’BUSNR-1 downto 0);)
_ifelse( CINTFU‘’FUNR‘’_RBUS‘’BUSNR, 1,
srcbus‘’BUSNR : in std_logic_vector(SRCSIZE‘’BUSNR-1 downto 0);)
_ifelse( CINTFU‘’FUNR‘’_TBUS‘’BUSNR, 1,
tllockreq‘’BUSNR : out std_logic; -- trigger socket local lock request)
_ifelse( CINTFU‘’FUNR‘’_OBUS‘’BUSNR,
1, ollockreq‘’BUSNR : out std_logic; -- operand socket local lock request)
_ifelse( CINTFU‘’FUNR‘’_RBUS‘’BUSNR,
1, rllockreq‘’BUSNR : out std_logic; -- result socket local lock request)
)--
texcreq : out std_logic; -- trigger socket exception request
oexcreq : out std_logic; -- operand socket exception request
rexcreq : out std_logic; -- result socket exception request
glock : in std_logic;
outvalid : out std_logic; -- valid std_logic output
valen : in std_logic; -- enable valid std_logic

ck : in std_logic;
rst : in std_logic
   );
end intfu‘’FUNR;

use work.sockets.all;
use work.internals.all;

architecture func of intfu‘’FUNR is
define({NRTBUSES}, translit(NINTFU\FUNR\_T,\))
define({NROBUSES}, translit(NINTFU\FUNR\_O,\))
define({NRRBUSES}, translit(NINTFU\FUNR\_R,\))

   for tsock:insock‘’NRTBUSES use entity WORK.insock‘’NRTBUSES‘’(ARCHSTYLE);
   for osock:insnop‘’NROBUSES use entity WORK.insnop‘’NROBUSES‘’(ARCHSTYLE);
   for rsock:outsnp‘’NRRBUSES use entity WORK.outsnp‘’NRRBUSES‘’(ARCHSTYLE);
   for fu_internals:intfui use entity WORK.intfui(ARCHSTYLE);
   for valsock:outval use entity WORK.outval(ARCHSTYLE);

   signal intern_tdata : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_topcode : std_logic_vector(LOG(eval(INTFUTS-1))-1 downto 0);
   signal intern_tcls : std_logic;
   signal intern_tbsy : std_logic;
   signal intern_toutsels : std_logic;
   signal intern_tdataact : std_logic;
   signal intern_tllockreq : std_logic_vector(NRTBUSES-1 downto 0);
   signal intern_tsquash : std_logic_vector(NRTBUSES-1 downto 0);

   signal intern_odata : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_ocls : std_logic;
   signal intern_obsy : std_logic;
   signal intern_ooutsels : std_logic;
   signal intern_odataact : std_logic;
   signal intern_ollockreq : std_logic_vector(NROBUSES-1 downto 0);
   signal intern_osquash : std_logic_vector(NROBUSES-1 downto 0);

   signal intern_rdata : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_ract : std_logic;
   signal intern_remp : std_logic;
   signal intern_rsocketcls : std_logic;
   signal intern_routsels : std_logic;
   signal intern_rllockreq : std_logic_vector(NRRBUSES-1 downto 0);
   signal intern_rsquash : std_logic_vector(NRRBUSES-1 downto 0);

begin
   tsock:insock‘’NRTBUSES
define({BSUBST}, translit(BINTFU\FUNR\_T,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NRTBUSES-1,
{ }ldstsize‘’BUSNR => _sub(DSTSIZE‘’BSUBST\BUSNR){,}
{ }lowid‘’BUSNR => _sub(AINTFU\FUNR\_TBUS‘’BSUBST\BUSNR){,}
){ }logopcodes => LOG(eval(INTFUTS-1))) ,\)
translit(
{      }port map(
FOR(BUSNR, 0, NRTBUSES-1,
{ }databus‘’BUSNR => databus‘’BSUBST\BUSNR{,}
{ }dstbus‘’BUSNR => dstbus‘’BSUBST\BUSNR{,}
){ }excreq => texcreq,
{ }llockreq => intern_tllockreq,
{ }squash => intern_tsquash,
```

```
{ }glock => glock,
{ }ck => ck,
{ }data => intern_tdata,
{ }opcode => intern_topcode,
{ }fucls => intern_tcls,
{ }fubsy => intern_tbsy,
{ }outsels => intern_toutsels,
{ }dataact => intern_tdataact );,\)

translit(
FOR(BUSNR, 0, NRTBUSES-1, {   }tllockreq`'BSUBST\BUSNR <= intern_tllockreq(BUSNR);
{   }intern_tsquash(BUSNR) <= squash`'BSUBST\BUSNR;
),\)

   osock:insnop`'NROBUSES
define({BSUBST}, translit(BINTFU\FUNR\_O,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NROBUSES-2,
{ }ldstsize`'BUSNR => _sub(DSTSIZE`'BSUBST\BUSNR){,}
{ }lowid`'BUSNR => _sub(AINTFU\FUNR\_OBUS`'BSUBST\BUSNR){,}
){ }ldstsize`'eval(NROBUSES-1) =>_sub(DSTSIZE`'BSUBST\eval(NROBUSES-1)){,}
{ }lowid`'eval(NROBUSES-1) => _sub(AINTFU\FUNR\_OBUS`'BSUBST\eval(NROBUSES-1))),\)
translit(
{      }port map(
FOR(BUSNR, 0, NROBUSES-1,
{ }databus`'BUSNR => databus`'BSUBST\BUSNR{,}
{ }dstbus`'BUSNR => dstbus`'BSUBST\BUSNR{,}
){ }excreq => oexcreq,
{ }llockreq => intern_ollockreq,
{ }squash => intern_osquash,
{ }glock => glock,
{ }ck => ck,
{ }data => intern_odata,
{ }fucls => intern_ocls,
{ }fubsy => intern_obsy,
{ }outsels => intern_ooutsels,
{ }dataact => intern_odataact );,\)

translit(
FOR(BUSNR, 0, NROBUSES-1, {   }ollockreq`'BSUBST\BUSNR <= intern_ollockreq(BUSNR);
{   }intern_osquash(BUSNR) <= squash`'BSUBST\BUSNR;
),\)

   rsock:outsnp`'NRRBUSES
define({BSUBST}, translit(BINTFU\FUNR\_R,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NRRBUSES-2,
{ }lsrcsize`'BUSNR => _sub(SRCSIZE`'BSUBST\BUSNR){,}
{ }lowid`'BUSNR => _sub(AINTFU\FUNR\_RBUS`'BSUBST\BUSNR){,}
){ }lsrcsize`'eval(NRRBUSES-1) =>_sub(SRCSIZE`'BSUBST\eval(NRRBUSES-1)){,}
{ }lowid`'eval(NRRBUSES-1) => _sub(AINTFU\FUNR\_RBUS`'BSUBST\eval(NRRBUSES-1))),\)
translit(
{      }port map(
FOR(BUSNR, 0, NRRBUSES-1,
{ }databus`'BUSNR => databus`'BSUBST\BUSNR{,}
{ }srcbus`'BUSNR => srcbus`'BSUBST\BUSNR{,}
){ }excreq => rexcreq,
{ }llockreq => intern_rllockreq,
{ }squash => intern_rsquash,
{ }glock => glock,
{ }ck => ck,
{ }data => intern_rdata,
{ }fuact => intern_ract,
{ }fuemp => intern_remp,
{ }socketcls => intern_rsocketcls,
{ }outsels => intern_routsels);,\)

translit(
FOR(BUSNR, 0, NRRBUSES-1, {   }rllockreq`'BSUBST\BUSNR <= intern_rllockreq(BUSNR);
{   }intern_rsquash(BUSNR) <= squash`'BSUBST\BUSNR;
),\)

   fu_internals:intfui
{      }port map(
{ }tdata => intern_tdata,
{ }topcode => intern_topcode,
{ }tcls => intern_tcls,
{ }tbsy => intern_tbsy,
{ }toutsels => intern_toutsels,
{ }tdataact => intern_tdataact,

{ }odata => intern_odata,
{ }ocls => intern_ocls,
{ }obsy => intern_obsy,
{ }ooutsels => intern_ooutsels,
```

```
{ }odataact => intern_odataact,

{ }rdata => intern_rdata,
{ }ract => intern_ract,
{ }remp => intern_remp,
{ }rsocketcls => intern_rsocketcls,
{ }routsels => intern_routsels,

{ }ck => ck,
{ }rst => rst);

   valsock:outval
      port map(
outvalid => outvalid,
vin => intern_ract,
enable => valen);

end func;
-------------------------------------------------------------------------------
-- Integer FU combinational logic
--
--`{ parameter dependencies: BUSWIDTH}'
-------------------------------------------------------------------------------

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

entity intfuc is
   port(
a : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
b : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
opc : in std_logic_vector(LOG(eval(INTFUTS-1))-1 downto 0);
y : out std_logic_vector(eval(BUSWIDTH-1) downto 0));
end intfuc;

architecture func of intfuc is

begin
   process(a, b, opc)
   begin
      case opc is
 when "0" => y <= conv_std_logic_vector(signed(a) + signed(b)
, y'length);
 when "1" => y <= conv_std_logic_vector(signed(a) - signed(b)
, y'length);
 when others => y <= a;
      end case;
   end process;
end func;
-------------------------------------------------------------------------------
-- Integer FU internals
--
--`{ parameter dependencies: BUSWIDTH INTFUDELAY}'
-------------------------------------------------------------------------------

library IEEE;
use IEEE.Std_Logic_1164.all;

entity intfui is
   port(
tdata : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
topcode : in std_logic_vector(LOG(eval(INTFUTS-1))-1 downto 0);
tcls : out std_logic;
tbsy : out std_logic;
toutsels : out std_logic;
tdataact : in std_logic;

odata : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
ocls : out std_logic;
obsy : out std_logic;
ooutsels : out std_logic;
odataact : in std_logic;

rdata : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
ract : out std_logic;
remp : out std_logic;
rsocketcls : in std_logic;
routsels : in std_logic;

ck : in std_logic;
rst : in std_logic);
end intfui;

use work.stagecontrol.all;
use work.combinational.all;
```

```
architecture func of intfui is

   for stagectl0:stagec use entity WORK.stagec(ARCHSTYLE);
   for stagectl1:stagecnb use entity WORK.stagecnb(ARCHSTYLE);
   for comblogic:intfuc use entity WORK.intfuc(ARCHSTYLE);

   signal intern_outcls : std_logic;
   signal intern_outactive : std_logic;
   signal intern_outempty : std_logic;
   signal intern_inbsy : std_logic;
   signal intern_incls : std_logic;

   signal stg0_enable : std_logic;
   signal stg1_enable : std_logic;

   signal wired_to_one : std_logic;
   signal wired_to_zero : std_logic;

   signal treg : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal oreg : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal rreg : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal opcreg : std_logic_vector(LOG(eval(INTFUTS-1))-1 downto 0);

begin
   stagectl0:stagec
generic map(stagedelay => INTFUDELAY)
port map(inactive  => tdataact,
                 inempty   => wired_to_one,
 inbsy     => intern_inbsy,
 incls     => intern_incls,

 ck        => ck,
 rst       => rst,
 socketcls => rsocketcls,
 enable    => stg0_enable,

 outactive => intern_outactive,
 outempty  => intern_outempty,
 outbsy    => wired_to_zero,
 outcls    => intern_outcls);
   stagectl1:stagecnb
port map(inactive  => intern_outactive,
                 inempty   => intern_outempty,
 incls     => intern_outcls,

 ck        => ck,
 rst       => rst,
 socketcls => rsocketcls,
 enable    => stg1_enable,

 outactive => ract,
 outempty  => remp,
 outcls    => wired_to_one);
   comblogic:intfuc
port map(a => treg,
 b => oreg,
 opc => opcreg,
 y => rreg);

   wired_to_one <= '1';
   wired_to_zero <= '0';

   tbsy <= intern_inbsy;
   obsy <= intern_inbsy;
   tcls <= intern_incls;
   ocls <= intern_incls;
   toutsels <= routsels;
   ooutsels <= routsels;

   process
   begin
      wait on ck until ck='1';
      if (stg1_enable='1') then
 rdata <= rreg;
      end if;
      if (stg0_enable='1') then
 treg <= tdata;
 opcreg <= topcode;
      end if;
      if (odataact='1') then
 oreg <= odata;
      end if;
   end process;

end func;
------------------------------------------------------------------------------
--  Load-Store FU FUNR
```

```
--
--`{ parameter dependencies: BUSWIDTH PARMOVES CLDSFU}'FUNR...
-------------------------------------------------------------------------------

library IEEE;
use IEEE.Std_Logic_1164.all;

entity ldsfu`'FUNR is
   port(
FOR(BUSNR, 0, PARMOVES-1,
--
_ifelse(_eval((CLDSFU`'FUNR`'_TBUS`'BUSNR || CLDSFU`'FUNR`'_OBUS`'BUSNR) &&
CLDSFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : inout std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval((CLDSFU`'FUNR`'_TBUS`'BUSNR || CLDSFU`'FUNR`'_OBUS`'BUSNR) &&
!CLDSFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval(!(CLDSFU`'FUNR`'_TBUS`'BUSNR || CLDSFU`'FUNR`'_OBUS`'BUSNR) &&
CLDSFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);)
_ifelse(_eval(CLDSFU`'FUNR`'_TBUS`'BUSNR || CLDSFU`'FUNR`'_OBUS`'BUSNR ||
CLDSFU`'FUNR`'_RBUS`'BUSNR), 1,
squash`'BUSNR : in std_logic;)
_ifelse(_eval(CLDSFU`'FUNR`'_TBUS`'BUSNR || CLDSFU`'FUNR`'_OBUS`'BUSNR), 1,
dstbus`'BUSNR : in std_logic_vector(DSTSIZE`'BUSNR-1 downto 0);)
_ifelse( CLDSFU`'FUNR`'_RBUS`'BUSNR, 1,
srcbus`'BUSNR : in std_logic_vector(SRCSIZE`'BUSNR-1 downto 0);)
_ifelse( CLDSFU`'FUNR`'_TBUS`'BUSNR, 1,
tllockreq`'BUSNR : out std_logic;)
_ifelse( CLDSFU`'FUNR`'_OBUS`'BUSNR,
1, ollockreq`'BUSNR : out std_logic;)
_ifelse( CLDSFU`'FUNR`'_RBUS`'BUSNR,
1, rllockreq`'BUSNR : out std_logic;)
)--
rexcreq : out std_logic;
glock : in std_logic;
outvalid : out std_logic;
valen : in std_logic;

dout : out std_logic_vector(BUSWIDTH-1 downto 0);
din : in std_logic_vector(BUSWIDTH-1 downto 0);
wnr : out std_logic;
da : out std_logic_vector(LDSADDRSIZE-1 downto 0);
das : out std_logic;
dl : out std_logic;

ck : in std_logic;
rst : in std_logic);
end ldsfu`'FUNR;

use work.sockets.all;
use work.internals.all;

architecture func of ldsfu`'FUNR is
define({NRTBUSES}, translit(NLDSFU\FUNR\_T,\))
define({NROBUSES}, translit(NLDSFU\FUNR\_O,\))
define({NRRBUSES}, translit(NLDSFU\FUNR\_R,\))

   for tsock:insock`'NRTBUSES use entity WORK.insock`'NRTBUSES`'(ARCHSTYLE);
   for osock:insnop`'NROBUSES use entity WORK.insnop`'NROBUSES`'(ARCHSTYLE);
   for rsock:outsnp`'NRRBUSES use entity WORK.outsnp`'NRRBUSES`'(ARCHSTYLE);
   for fu_internals:ldsfui use entity WORK.ldsfui(ARCHSTYLE);
   for valsock:outval use entity WORK.outval(ARCHSTYLE);

   signal intern_tdata : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_topcode : std_logic_vector(LOG(eval(LDSFUTS-1))-1 downto 0);
   signal intern_tcls : std_logic;
   signal intern_tbsy : std_logic;
   signal intern_toutsels : std_logic;
   signal intern_tdataact : std_logic;
   signal intern_tllockreq : std_logic_vector(NRTBUSES-1 downto 0);
   signal intern_tsquash : std_logic_vector(NRTBUSES-1 downto 0);

   signal intern_odata : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_ocls : std_logic;
   signal intern_obsy : std_logic;
   signal intern_ooutsels : std_logic;
   signal intern_odataact : std_logic;
   signal intern_ollockreq : std_logic_vector(NROBUSES-1 downto 0);
   signal intern_osquash : std_logic_vector(NROBUSES-1 downto 0);

   signal intern_rdata : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_ract : std_logic;
   signal intern_remp : std_logic;
   signal intern_rsocketcls : std_logic;
   signal intern_routsels : std_logic;
   signal intern_rllockreq : std_logic_vector(NRRBUSES-1 downto 0);
```

```
   signal intern_rsquash : std_logic_vector(NRRBUSES-1 downto 0);

begin
   tsock:insock‘’NRTBUSES
define({BSUBST}, translit(BLDSFU\FUNR\_T,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NRTBUSES-1,
{ }ldstsize‘’BUSNR => _sub(DSTSIZE‘’BSUBST\BUSNR){,}
{ }lowid‘’BUSNR => _sub(ALDSFU\FUNR\_TBUS‘’BSUBST\BUSNR){,}
){ }logopcodes => LOG(eval(LDSFUTS-1))) ,\)
translit(
{      }port map(
FOR(BUSNR, 0, NRTBUSES-1,
{ }databus‘’BUSNR => databus‘’BSUBST\BUSNR{,}
{ }dstbus‘’BUSNR => dstbus‘’BSUBST\BUSNR{,}
)--{ }excreq => texcreq,
{ }llockreq => intern_tllockreq,
{ }squash => intern_tsquash,
{ }glock => glock,
{ }ck => ck,
{ }data => intern_tdata,
{ }opcode => intern_topcode,
{ }fucls => intern_tcls,
{ }fubsy => intern_tbsy,
{ }outsels => intern_toutsels,
{ }dataact => intern_tdataact );,\)

translit(
FOR(BUSNR, 0, NRTBUSES-1, {   }tllockreq‘’BSUBST\BUSNR <= intern_tllockreq(BUSNR);
{   }intern_tsquash(BUSNR) <= squash‘’BSUBST\BUSNR;
),\)
   osock:insnop‘’NROBUSES
define({BSUBST}, translit(BLDSFU\FUNR\_O,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NROBUSES-2,
{ }ldstsize‘’BUSNR => _sub(DSTSIZE‘’BSUBST\BUSNR){,}
{ }lowid‘’BUSNR => _sub(ALDSFU\FUNR\_OBUS‘’BSUBST\BUSNR){,}
){ }ldstsize‘’eval(NROBUSES-1) =>_sub(DSTSIZE‘’BSUBST\eval(NROBUSES-1)){,}
{ }lowid‘’eval(NROBUSES-1) => _sub(ALDSFU\FUNR\_OBUS‘’BSUBST\eval(NROBUSES-1))),\)
translit(
{      }port map(
FOR(BUSNR, 0, NROBUSES-1,
{ }databus‘’BUSNR => databus‘’BSUBST\BUSNR{,}
{ }dstbus‘’BUSNR => dstbus‘’BSUBST\BUSNR{,}
)--{ }excreq => oexcreq,
{ }llockreq => intern_ollockreq,
{ }squash => intern_osquash,
{ }glock => glock,
{ }ck => ck,
{ }data => intern_odata,
{ }fucls => intern_ocls,
{ }fubsy => intern_obsy,
{ }outsels => intern_ooutsels,
{ }dataact => intern_odataact );,\)

translit(
FOR(BUSNR, 0, NROBUSES-1, {   }ollockreq‘’BSUBST\BUSNR <= intern_ollockreq(BUSNR);
{   }intern_osquash(BUSNR) <= squash‘’BSUBST\BUSNR;
),\)
   rsock:outsnp‘’NRRBUSES
define({BSUBST}, translit(BLDSFU\FUNR\_R,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NRRBUSES-2,
{ }lsrcsize‘’BUSNR => _sub(SRCSIZE‘’BSUBST\BUSNR){,}
{ }lowid‘’BUSNR => _sub(ALDSFU\FUNR\_RBUS‘’BSUBST\BUSNR){,}
){ }lsrcsize‘’eval(NRRBUSES-1) =>_sub(SRCSIZE‘’BSUBST\eval(NRRBUSES-1)){,}
{ }lowid‘’eval(NRRBUSES-1) => _sub(ALDSFU\FUNR\_RBUS‘’BSUBST\eval(NRRBUSES-1))),\)
translit(
{      }port map(
FOR(BUSNR, 0, NRRBUSES-1,
{ }databus‘’BUSNR => databus‘’BSUBST\BUSNR{,}
{ }srcbus‘’BUSNR => srcbus‘’BSUBST\BUSNR{,}
){ }excreq => rexcreq,
{ }llockreq => intern_rllockreq,
{ }squash => intern_rsquash,
{ }glock => glock,
{ }ck => ck,
{ }data => intern_rdata,
{ }fuact => intern_ract,
{ }fuemp => intern_remp,
{ }socketcls => intern_rsocketcls,
{ }outsels => intern_routsels);,\)

translit(
```

```
FOR(BUSNR, 0, NRRBUSES-1, {   }rllockreq`'BSUBST\BUSNR <= intern_rllockreq(BUSNR);
{   }intern_rsquash(BUSNR) <= squash`'BSUBST\BUSNR;
),\)

   fu_internals:ldsfui
{      }port map(
{ }tdata => intern_tdata,
{ }topcode => intern_topcode,
{ }tcls => intern_tcls,
{ }tbsy => intern_tbsy,
{ }toutsels => intern_toutsels,
{ }tdataact => intern_tdataact,

{ }odata => intern_odata,
{ }ocls => intern_ocls,
{ }obsy => intern_obsy,
{ }ooutsels => intern_ooutsels,
{ }odataact => intern_odataact,

{ }rdata => intern_rdata,
{ }ract => intern_ract,
{ }remp => intern_remp,
{ }rsocketcls => intern_rsocketcls,
{ }routsels => intern_routsels,

{ }dout => dout,
{ }din => din,
{ }wnr => wnr,
{ }da => da,
{ }das => das,
{ }dl => dl,

{ }ck => ck,
{ }rst => rst);

   valsock:outval
      port map(
outvalid => outvalid,
vin => intern_ract,
enable => valen);

end func;
-------------------------------------------------------------------------------
--  Load-Store FU combinational logic
--
--`{ parameter dependencies: LDSOLOCKFULL}'
-------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;

entity ldsfuc is
   port(st0bsy : in std_logic;
wnr : in std_logic;
wnro : out std_logic;
obsy : out std_logic;
st0outactive : in std_logic;
st1inactive : out std_logic;
st0outempty : in std_logic;
st1inempty : out std_logic);
end ldsfuc;


architecture func of ldsfuc is
begin
   wnro <= not wnr;
ifdef({LDSOLOCKFULL}, {   }obsy <= st0bsy and wnr;, {   }obsy <= st0bsy;)
   st1inactive <= st0outactive and not wnr;
   st1inempty <= st0outempty or wnr;
end func;
-------------------------------------------------------------------------------
--  Load-Store FU Internals
--
--  opcodes:
--  0  load
--  1  store
--
--  T: address
--  O: store data
--  R: load data
--
--`{ parameter dependencies: BUSWIDTH LDSADDRSIZE LDSFUDELAY}'
-------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity ldsfui is
```

```
   port(tdata : in std_logic_vector(BUSWIDTH-1 downto 0);
        topcode : in std_logic_vector(LOG(eval(LDSFUTS-1))-1 downto 0);
        tcls : out std_logic;
        tbsy : out std_logic;
        toutsels : out std_logic;
        tdataact : in std_logic;

        odata : in std_logic_vector(BUSWIDTH-1 downto 0);
        ocls : out std_logic;
        obsy : out std_logic;
        ooutsels : out std_logic;
        odataact : in std_logic;

        rdata : out std_logic_vector(BUSWIDTH-1 downto 0);
        ract : out std_logic;
        remp : out std_logic;
        rsocketcls : in std_logic;
        routsels : in std_logic;

dout : out std_logic_vector(BUSWIDTH-1 downto 0);
din : in std_logic_vector(BUSWIDTH-1 downto 0);
wnr : out std_logic;
da : out std_logic_vector(LDSADDRSIZE-1 downto 0);
das : out std_logic;
dl : out std_logic;

        ck : in std_logic;
        rst : in std_logic);
end ldsfui;

use work.stagecontrol.all;
use work.combinational.all;

architecture func of ldsfui is

   for stagectl0:stagec use entity WORK.stagec(ARCHSTYLE);
   for stagectl1:stagecnb use entity WORK.stagecnb(ARCHSTYLE);
   for comblogic:ldsfuc use entity WORK.ldsfuc(ARCHSTYLE);

   signal stg0_outcls : std_logic;
   signal stg0_outactive : std_logic;
   signal stg0_outempty : std_logic;
   signal stg0_inbsy : std_logic;
   signal stg0_incls : std_logic;

   signal stg1_inactive : std_logic;
   signal stg1_inempty : std_logic;
   signal stg1_incls : std_logic;

   signal stg0_enable : std_logic;
   signal stg1_enable : std_logic;

   signal wired_to_one : std_logic;
   signal wired_to_zero : std_logic;

   signal treg : std_logic_vector(LDSADDRSIZE-1 downto 0);
   signal oreg : std_logic_vector(BUSWIDTH-1 downto 0);
   signal rreg : std_logic_vector(BUSWIDTH-1 downto 0);
   signal opcreg : std_logic_vector(LOG(eval(LDSFUTS-1))-1 downto 0);

begin
   stagectl0:stagec
        generic map(stagedelay => LDSFUDELAY)
        port map(inactive  => tdataact,
                 inempty   => wired_to_one,
                 inbsy     => stg0_inbsy,
                 incls     => stg0_incls,

                 ck        => ck,
                 rst       => rst,
                 socketcls => wired_to_zero,
                 enable    => stg0_enable,

                 outactive => stg0_outactive,
                 outempty  => stg0_outempty,
                 outbsy    => wired_to_zero,
                 outcls    => wired_to_zero);

   stagectl1:stagecnb
        port map(inactive  => stg1_inactive,
                 inempty   => stg1_inempty,
                 incls     => stg1_incls,

                 ck        => ck,
                 rst       => rst,
                 socketcls => rsocketcls,
                 enable    => stg1_enable,
```

```
                 outactive => ract,
                 outempty  => remp,
                 outcls    => wired_to_one);

   comblogic:ldsfuc
port map(st0bsy => stg0_inbsy,
 wnr => opcreg(0),
 wnro => wnr,
 obsy => obsy,
 st0outactive => stg0_outactive,
 st1inactive => stg1_inactive,
 st0outempty => stg0_outempty,
 st1inempty => stg1_inempty);

   wired_to_one <= '1';
   wired_to_zero <= '0';

   rdata <= rreg;
   dout <= oreg;
   da <= treg;

   dl <= stg0_outactive;

   tcls <= '0';
   tbsy <= stg0_inbsy;
   toutsels <= '1';

   ocls <= '0';
   ooutsels <= '1';

   process
   begin
      wait on ck until ck='1';
      das <= stg0_enable;
      if (stg1_enable='1') then
 rreg <= din;
      end if;
      if (stg0_enable='1') then
 treg <= tdata(LDSADDRSIZE-1 downto 0);
 opcreg <= topcode;
      end if;
      if (odataact='1') then
 oreg <= odata;
      end if;
   end process;

end func;
------------------------------------------------------------------------------
-- Logic FU FUNR
--
--`{ parameter dependencies: BUSWIDTH PARMOVES CLOGFU}'FUNR...
------------------------------------------------------------------------------

library IEEE;
use IEEE.Std_Logic_1164.all;

entity logfu`'FUNR is
   port(
FOR(BUSNR, 0, PARMOVES-1,
--
_ifelse(_eval((CLOGFU`'FUNR`'_TBUS`'BUSNR || CLOGFU`'FUNR`'_OBUS`'BUSNR) &&
CLOGFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : inout std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval((CLOGFU`'FUNR`'_TBUS`'BUSNR || CLOGFU`'FUNR`'_OBUS`'BUSNR) &&
!CLOGFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval(!(CLOGFU`'FUNR`'_TBUS`'BUSNR || CLOGFU`'FUNR`'_OBUS`'BUSNR) &&
CLOGFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);)
_ifelse(_eval(CLOGFU`'FUNR`'_TBUS`'BUSNR || CLOGFU`'FUNR`'_OBUS`'BUSNR ||
CLOGFU`'FUNR`'_RBUS`'BUSNR), 1,
squash`'BUSNR : in std_logic;)
_ifelse(_eval(CLOGFU`'FUNR`'_TBUS`'BUSNR || CLOGFU`'FUNR`'_OBUS`'BUSNR), 1,
dstbus`'BUSNR : in std_logic_vector(DSTSIZE`'BUSNR-1 downto 0);)
_ifelse( CLOGFU`'FUNR`'_RBUS`'BUSNR, 1,
srcbus`'BUSNR : in std_logic_vector(SRCSIZE`'BUSNR-1 downto 0);)
_ifelse( CLOGFU`'FUNR`'_RBUS`'BUSNR,
1, rllockreq`'BUSNR : out std_logic; -- result socket local lock request)
)--
texcreq : out std_logic; -- trigger socket exception request
oexcreq : out std_logic; -- operand socket exception request
rexcreq : out std_logic; -- result socket exception request
glock : in std_logic;
outvalid : out std_logic; -- valid std_logic output
valen : in std_logic; -- enable valid std_logic
```

```
ck : in std_logic;
rst : in std_logic
   );
end logfu‘’FUNR;


use work.sockets.all;
use work.internals.all;

architecture func of logfu‘’FUNR is
define({NRTBUSES}, translit(NLOGFU\FUNR\_T,\))
define({NROBUSES}, translit(NLOGFU\FUNR\_O,\))
define({NRRBUSES}, translit(NLOGFU\FUNR\_R,\))

   for tsock:insknb‘’NRTBUSES use entity WORK.insknb‘’NRTBUSES‘’(ARCHSTYLE);
   for osock:inspnb‘’NROBUSES use entity WORK.inspnb‘’NROBUSES‘’(ARCHSTYLE);
   for rsock:outsnp‘’NRRBUSES use entity WORK.outsnp‘’NRRBUSES‘’(ARCHSTYLE);
   for fu_internals:logfui use entity WORK.logfui(ARCHSTYLE);
   for valsock:outval use entity WORK.outval(ARCHSTYLE);

   signal intern_tdata : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_topcode : std_logic_vector(LOG(eval(LOGFUTS-1))-1 downto 0);
   signal intern_tcls : std_logic;
   signal intern_toutsels : std_logic;
   signal intern_tdataact : std_logic;
   signal intern_tsquash : std_logic_vector(NRTBUSES-1 downto 0);

   signal intern_odata : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_ocls : std_logic;
   signal intern_ooutsels : std_logic;
   signal intern_odataact : std_logic;
   signal intern_osquash : std_logic_vector(NROBUSES-1 downto 0);

   signal intern_rdata : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_ract : std_logic;
   signal intern_remp : std_logic;
   signal intern_rsocketcls : std_logic;
   signal intern_routsels : std_logic;
   signal intern_rllockreq : std_logic_vector(NRRBUSES-1 downto 0);
   signal intern_rsquash : std_logic_vector(NRRBUSES-1 downto 0);

begin
   tsock:insknb‘’NRTBUSES
define({BSUBST}, translit(BLOGFU\FUNR\_T,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NRTBUSES-1,
{ }ldstsize‘’BUSNR => _sub(DSTSIZE‘’BSUBST\BUSNR){,}
{ }lowid‘’BUSNR => _sub(ALOGFU\FUNR\_TBUS‘’BSUBST\BUSNR){,}
){ }logopcodes => LOG(eval(LOGFUTS-1))) ,\)
translit(
{      }port map(
FOR(BUSNR, 0, NRTBUSES-1,
{ }databus‘’BUSNR => databus‘’BSUBST\BUSNR{,}
{ }dstbus‘’BUSNR => dstbus‘’BSUBST\BUSNR{,}
){ }excreq => texcreq,
{ }squash => intern_tsquash,
{ }glock => glock,
{ }ck => ck,
{ }data => intern_tdata,
{ }opcode => intern_topcode,
{ }fucls => intern_tcls,
{ }outsels => intern_toutsels,
{ }dataact => intern_tdataact );,\)

translit(
FOR(BUSNR, 0, NRTBUSES-1, {   }intern_tsquash(BUSNR) <= squash‘’BSUBST\BUSNR;
),\)

   osock:inspnb‘’NROBUSES
define({BSUBST}, translit(BLOGFU\FUNR\_O,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NROBUSES-2,
{ }ldstsize‘’BUSNR => _sub(DSTSIZE‘’BSUBST\BUSNR){,}
{ }lowid‘’BUSNR => _sub(ALOGFU\FUNR\_OBUS‘’BSUBST\BUSNR){,}
){ }ldstsize‘’eval(NROBUSES-1) =>_sub(DSTSIZE‘’BSUBST\eval(NROBUSES-1)){,}
{ }lowid‘’eval(NROBUSES-1) => _sub(ALOGFU\FUNR\_OBUS‘’BSUBST\eval(NROBUSES-1))),\)
translit(
{      }port map(
FOR(BUSNR, 0, NROBUSES-1,
{ }databus‘’BUSNR => databus‘’BSUBST\BUSNR{,}
{ }dstbus‘’BUSNR => dstbus‘’BSUBST\BUSNR{,}
){ }excreq => oexcreq,
{ }squash => intern_osquash,
{ }glock => glock,
{ }ck => ck,
```

```
{ }data => intern_odata,
{ }fucls => intern_ocls,
{ }outsels => intern_ooutsels,
{ }dataact => intern_odataact );,\)

translit(
FOR(BUSNR, 0, NROBUSES-1, {   }intern_osquash(BUSNR) <= squash`'BSUBST\BUSNR;
),\)

   rsock:outsnp`'NRRBUSES
define({BSUBST}, translit(BLOGFU\FUNR\_R,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NRRBUSES-2,
{ }lsrcsize`'BUSNR => _sub(SRCSIZE`'BSUBST\BUSNR){,}
{ }lowid`'BUSNR => _sub(ALOGFU\FUNR\_RBUS`'BSUBST\BUSNR){,}
){ }lsrcsize`'eval(NRRBUSES-1) =>_sub(SRCSIZE`'BSUBST\eval(NRRBUSES-1)){,}
{ }lowid`'eval(NRRBUSES-1) => _sub(ALOGFU\FUNR\_RBUS`'BSUBST\eval(NRRBUSES-1))),\)
translit(
{      }port map(
FOR(BUSNR, 0, NRRBUSES-1,
{ }databus`'BUSNR => databus`'BSUBST\BUSNR{,}
{ }srcbus`'BUSNR => srcbus`'BSUBST\BUSNR{,}
){ }excreq => rexcreq,
{ }llockreq => intern_rllockreq,
{ }squash => intern_rsquash,
{ }glock => glock,
{ }ck => ck,
{ }data => intern_rdata,
{ }fuact => intern_ract,
{ }fuemp => intern_remp,
{ }socketcls => intern_rsocketcls,
{ }outsels => intern_routsels);,\)

translit(
FOR(BUSNR, 0, NRRBUSES-1, {   }rllockreq`'BSUBST\BUSNR <= intern_rllockreq(BUSNR);
{   }intern_rsquash(BUSNR) <= squash`'BSUBST\BUSNR;
),\)

   fu_internals:logfui
{      }port map(
{ }tdata => intern_tdata,
{ }topcode => intern_topcode,
{ }tcls => intern_tcls,
{ }toutsels => intern_toutsels,
{ }tdataact => intern_tdataact,

{ }odata => intern_odata,
{ }ocls => intern_ocls,
{ }ooutsels => intern_ooutsels,
{ }odataact => intern_odataact,

{ }rdata => intern_rdata,
{ }ract => intern_ract,
{ }remp => intern_remp,
{ }rsocketcls => intern_rsocketcls,
{ }routsels => intern_routsels,

{ }ck => ck,
{ }rst => rst);

   valsock:outval
      port map(
outvalid => outvalid,
vin => intern_ract,
enable => valen);

end func;
------------------------------------------------------------------------------
-- Logic FU combinational logic
--
--`{ parameter dependencies: BUSWIDTH}'
------------------------------------------------------------------------------
library IEEE;
use IEEE.Std_Logic_1164.all;

entity logfuc is
   port(a : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
b : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
opc : in std_logic_vector(LOG(eval(LOGFUTS-1))-1 downto 0);
y : out std_logic_vector(eval(BUSWIDTH-1) downto 0));
end logfuc;

architecture func of logfuc is
begin
   process(a, b, opc)
   begin
```

```
      case opc is
 when "00" => y <= a and b;
 when "01" => y <= a or b;
 when "10" => y <= a xor b;
 when others => y <= "FOR(CNT,1,BUSWIDTH,-)";
      end case;
   end process;
end func;
------------------------------------------------------------------------------
-- Logic FU internals
--
--   opcodes:
--  0  and
--  1  or
--  2  exor
--  3  <undefined>
--
--`{ parameter dependencies: BUSWIDTH}'
------------------------------------------------------------------------------

library IEEE;
use IEEE.Std_Logic_1164.all;

entity logfui is
   port(
tdata : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
topcode : in std_logic_vector(LOG(eval(LOGFUTS-1))-1 downto 0);
tcls : out std_logic;
toutsels : out std_logic;
tdataact : in std_logic;

odata : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
ocls : out std_logic;
ooutsels : out std_logic;
odataact : in std_logic;

rdata : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
ract : out std_logic;
remp : out std_logic;
rsocketcls : in std_logic;
routsels : in std_logic;

ck : in std_logic;
rst : in std_logic);
end logfui;

use work.stagecontrol.all;
use work.combinational.all;

architecture func of logfui is

   for stagectl0:stagecnb use entity WORK.stagecnb(ARCHSTYLE);
   for stagectl1:stagecnb use entity WORK.stagecnb(ARCHSTYLE);
   for comblogic:logfuc use entity WORK.logfuc(ARCHSTYLE);

   signal intern_outcls : std_logic;
   signal intern_outactive : std_logic;
   signal intern_outempty : std_logic;
   signal intern_incls : std_logic;

   signal stg0_enable : std_logic;
   signal stg1_enable : std_logic;

   signal wired_to_one : std_logic;
   signal wired_to_zero : std_logic;

   signal treg : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal oreg : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal rreg : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal opcreg : std_logic_vector(LOG(eval(LOGFUTS-1))-1 downto 0);

begin
   stagectl0:stagecnb
port map(inactive  => tdataact,
                 inempty   => wired_to_one,
 incls     => intern_incls,

 ck        => ck,
 rst       => rst,
 socketcls => rsocketcls,
 enable    => stg0_enable,

 outactive => intern_outactive,
 outempty  => intern_outempty,
 outcls    => intern_outcls);
   stagectl1:stagecnb
port map(inactive  => intern_outactive,
```

```
                inempty   => intern_outempty,
 incls     => intern_outcls,

 ck        => ck,
 rst       => rst,
 socketcls => rsocketcls,
 enable    => stg1_enable,

 outactive => ract,
 outempty  => remp,
 outcls    => wired_to_one);
   comblogic:logfuc
port map(a => treg,
 b => oreg,
 opc => opcreg,
 y => rreg);

   wired_to_one <= '1';
   wired_to_zero <= '0';

   tcls <= intern_incls;
   ocls <= intern_incls;
   toutsels <= routsels;
   ooutsels <= routsels;

   process
   begin
      wait on ck until ck='1';
      if (stg1_enable='1') then
 rdata <= rreg;
      end if;
      if (stg0_enable='1') then
 treg <= tdata;
 opcreg <= topcode;
      end if;
      if (odataact='1') then
 oreg <= odata;
      end if;
   end process;
end func;
------------------------------------------------------------------------------
-- Multiply FU FUNR
--
--`{ parameter dependencies: BUSWIDTH PARMOVES CMULFU}'FUNR
------------------------------------------------------------------------------

library IEEE;
use IEEE.Std_Logic_1164.all;

entity mulfu`'FUNR is
   port(
FOR(BUSNR, 0, PARMOVES-1,
--
_ifelse(_eval((CMULFU`'FUNR`'_TBUS`'BUSNR || CMULFU`'FUNR`'_OBUS`'BUSNR) &&
(CMULFU`'FUNR`'_R1BUS`'BUSNR || CMULFU`'FUNR`'_R2BUS`'BUSNR)), 1,
databus`'BUSNR : inout std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval((CMULFU`'FUNR`'_TBUS`'BUSNR || CMULFU`'FUNR`'_OBUS`'BUSNR) &&
!(CMULFU`'FUNR`'_R1BUS`'BUSNR || CMULFU`'FUNR`'_R2BUS`'BUSNR)), 1,
databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval(!(CMULFU`'FUNR`'_TBUS`'BUSNR || CMULFU`'FUNR`'_OBUS`'BUSNR) &&
(CMULFU`'FUNR`'_R1BUS`'BUSNR || CMULFU`'FUNR`'_R2BUS`'BUSNR)), 1,
databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);)
_ifelse(_eval(CMULFU`'FUNR`'_TBUS`'BUSNR || CMULFU`'FUNR`'_OBUS`'BUSNR ||
CMULFU`'FUNR`'_R1BUS`'BUSNR || CMULFU`'FUNR`'_R2BUS`'BUSNR), 1,
squash`'BUSNR : in std_logic;)
_ifelse(_eval(CMULFU`'FUNR`'_TBUS`'BUSNR || CMULFU`'FUNR`'_OBUS`'BUSNR), 1,
dstbus`'BUSNR : in std_logic_vector(DSTSIZE`'BUSNR-1 downto 0);)
_ifelse(_eval(CMULFU`'FUNR`'_R1BUS`'BUSNR || CMULFU`'FUNR`'_R2BUS`'BUSNR), 1,
srcbus`'BUSNR : in std_logic_vector(SRCSIZE`'BUSNR-1 downto 0);)
_ifelse( CMULFU`'FUNR`'_TBUS`'BUSNR, 1,
tllockreq`'BUSNR : out std_logic;)
_ifelse( CMULFU`'FUNR`'_OBUS`'BUSNR,
1, ollockreq`'BUSNR : out std_logic;)
_ifelse( CMULFU`'FUNR`'_R1BUS`'BUSNR,
1, r1llockreq`'BUSNR : out std_logic;)
_ifelse( CMULFU`'FUNR`'_R2BUS`'BUSNR,
1, r2llockreq`'BUSNR : out std_logic;)
)--
texcreq : out std_logic;
oexcreq : out std_logic;
r1excreq : out std_logic;
r2excreq : out std_logic;
glock : in std_logic;

outvalid : out std_logic;
valen : in std_logic;
```

```
ck : in std_logic;
rst : in std_logic);
end mulfu‘’FUNR;

use work.sockets.all;
use work.internals.all;

architecture func of mulfu‘’FUNR is
define({NRTBUSES}, translit(NMULFU\FUNR\_T,\))
define({NROBUSES}, translit(NMULFU\FUNR\_O,\))
define({NRR1BUSES}, translit(NMULFU\FUNR\_R1,\))
define({NRR2BUSES}, translit(NMULFU\FUNR\_R2,\))

   for tsock:insnop‘’NRTBUSES use entity WORK.insnop‘’NRTBUSES‘’(ARCHSTYLE);
   for osock:insnop‘’NROBUSES use entity WORK.insnop‘’NROBUSES‘’(ARCHSTYLE);
   for r1sock:outsnp‘’NRR1BUSES use entity WORK.outsnp‘’NRR1BUSES‘’(ARCHSTYLE);
   for r2sock:outsnp‘’NRR2BUSES use entity WORK.outsnp‘’NRR2BUSES‘’(ARCHSTYLE);
   for fu_internals:mulfui use entity WORK.mulfui(ARCHSTYLE);
   for valsock:outval use entity WORK.outval(ARCHSTYLE);

   signal intern_tdata : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_tcls : std_logic;
   signal intern_tbsy : std_logic;
   signal intern_toutsels : std_logic;
   signal intern_tdataact : std_logic;
   signal intern_tllockreq : std_logic_vector(NRTBUSES-1 downto 0);
   signal intern_tsquash : std_logic_vector(NRTBUSES-1 downto 0);

   signal intern_odata : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_ocls : std_logic;
   signal intern_obsy : std_logic;
   signal intern_ooutsels : std_logic;
   signal intern_odataact : std_logic;
   signal intern_ollockreq : std_logic_vector(NROBUSES-1 downto 0);
   signal intern_osquash : std_logic_vector(NROBUSES-1 downto 0);

   signal intern_r1data : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_r1act : std_logic;
   signal intern_r1emp : std_logic;
   signal intern_r1socketcls : std_logic;
   signal intern_r1outsels : std_logic;
   signal intern_r1llockreq : std_logic_vector(NRR1BUSES-1 downto 0);
   signal intern_r1squash : std_logic_vector(NRR1BUSES-1 downto 0);

   signal intern_r2data : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_r2act : std_logic;
   signal intern_r2emp : std_logic;
   signal intern_r2socketcls : std_logic;
   signal intern_r2outsels : std_logic;
   signal intern_r2llockreq : std_logic_vector(NRR2BUSES-1 downto 0);
   signal intern_r2squash : std_logic_vector(NRR2BUSES-1 downto 0);

begin
   tsock:insnop‘’NRTBUSES
define({BSUBST}, translit(BMULFU\FUNR\_T,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NRTBUSES-2,
{ }ldstsize‘’BUSNR => _sub(DSTSIZE‘’BSUBST\BUSNR){,}
{ }lowid‘’BUSNR => _sub(AMULFU\FUNR\_TBUS‘’BSUBST\BUSNR){,}
){ }ldstsize‘’eval(NRTBUSES-1) =>_sub(DSTSIZE‘’BSUBST\eval(NRTBUSES-1)){,}
{ }lowid‘’eval(NRTBUSES-1) => _sub(AMULFU\FUNR\_TBUS‘’BSUBST\eval(NRTBUSES-1))),\)
translit(
{      }port map(
FOR(BUSNR, 0, NRTBUSES-1,
{ }databus‘’BUSNR => databus‘’BSUBST\BUSNR{,}
{ }dstbus‘’BUSNR => dstbus‘’BSUBST\BUSNR{,}
){ }excreq => texcreq,
{ }llockreq => intern_tllockreq,
{ }squash => intern_tsquash,
{ }glock => glock,
{ }ck => ck,
{ }data => intern_tdata,
{ }fucls => intern_tcls,
{ }fubsy => intern_tbsy,
{ }outsels => intern_toutsels,
{ }dataact => intern_tdataact );,\)

translit(
FOR(BUSNR, 0, NRTBUSES-1, {   }tllockreq‘’BSUBST\BUSNR <= intern_tllockreq(BUSNR);
{   }intern_tsquash(BUSNR) <= squash‘’BSUBST\BUSNR;
),\)

   osock:insnop‘’NROBUSES
define({BSUBST}, translit(BMULFU\FUNR\_O,\))
translit(
{      }generic map(
```

```
FOR(BUSNR, 0, NROBUSES-2,
{ }ldstsize`'BUSNR => _sub(DSTSIZE`'BSUBST\BUSNR){,}
{ }lowid`'BUSNR => _sub(AMULFU\FUNR\_OBUS`'BSUBST\BUSNR){,}
){ }ldstsize`'eval(NROBUSES-1) =>_sub(DSTSIZE`'BSUBST\eval(NROBUSES-1)){,}
{ }lowid`'eval(NROBUSES-1) => _sub(AMULFU\FUNR\_OBUS`'BSUBST\eval(NROBUSES-1))),\)
translit(
{      }port map(
FOR(BUSNR, 0, NROBUSES-1,
{ }databus`'BUSNR => databus`'BSUBST\BUSNR{,}
{ }dstbus`'BUSNR => dstbus`'BSUBST\BUSNR{,}
){ }excreq => oexcreq,
{ }llockreq => intern_ollockreq,
{ }squash => intern_osquash,
{ }glock => glock,
{ }ck => ck,
{ }data => intern_odata,
{ }fucls => intern_ocls,
{ }fubsy => intern_obsy,
{ }outsels => intern_ooutsels,
{ }dataact => intern_odataact );,\)

translit(
FOR(BUSNR, 0, NROBUSES-1, {    }ollockreq`'BSUBST\BUSNR <= intern_ollockreq(BUSNR);
{    }intern_osquash(BUSNR) <= squash`'BSUBST\BUSNR;
),\)

   r1sock:outsnp`'NRR1BUSES
define({BSUBST}, translit(BMULFU\FUNR\_R1,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NRR1BUSES-2,
{ }lsrcsize`'BUSNR => _sub(SRCSIZE`'BSUBST\BUSNR){,}
{ }lowid`'BUSNR => _sub(AMULFU\FUNR\_R1BUS`'BSUBST\BUSNR){,}
){ }lsrcsize`'eval(NRR1BUSES-1) =>_sub(SRCSIZE`'BSUBST\eval(NRR1BUSES-1)){,}
{ }lowid`'eval(NRR1BUSES-1) => _sub(AMULFU\FUNR\_R1BUS`'BSUBST\eval(NRR1BUSES-1))),\)
translit(
{      }port map(
FOR(BUSNR, 0, NRR1BUSES-1,
{ }databus`'BUSNR => databus`'BSUBST\BUSNR{,}
{ }srcbus`'BUSNR => srcbus`'BSUBST\BUSNR{,}
){ }excreq => r1excreq,
{ }llockreq => intern_r1llockreq,
{ }squash => intern_r1squash,
{ }glock => glock,
{ }ck => ck,
{ }data => intern_r1data,
{ }fuact => intern_r1act,
{ }fuemp => intern_r1emp,
{ }socketcls => intern_r1socketcls,
{ }outsels => intern_r1outsels);,\)

translit(
FOR(BUSNR, 0, NRR1BUSES-1, {    }r1llockreq`'BSUBST\BUSNR <= intern_r1llockreq(BUSNR);
{    }intern_r1squash(BUSNR) <= squash`'BSUBST\BUSNR;
),\)

   r2sock:outsnp`'NRR2BUSES
define({BSUBST}, translit(BMULFU\FUNR\_R2,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NRR2BUSES-2,
{ }lsrcsize`'BUSNR => _sub(SRCSIZE`'BSUBST\BUSNR){,}
{ }lowid`'BUSNR => _sub(AMULFU\FUNR\_R2BUS`'BSUBST\BUSNR){,}
){ }lsrcsize`'eval(NRR2BUSES-1) =>_sub(SRCSIZE`'BSUBST\eval(NRR2BUSES-1)){,}
{ }lowid`'eval(NRR2BUSES-1) => _sub(AMULFU\FUNR\_R2BUS`'BSUBST\eval(NRR2BUSES-1))),\)
translit(
{      }port map(
FOR(BUSNR, 0, NRR2BUSES-1,
{ }databus`'BUSNR => databus`'BSUBST\BUSNR{,}
{ }srcbus`'BUSNR => srcbus`'BSUBST\BUSNR{,}
){ }excreq => r2excreq,
{ }llockreq => intern_r2llockreq,
{ }squash => intern_r2squash,
{ }glock => glock,
{ }ck => ck,
{ }data => intern_r2data,
{ }fuact => intern_r2act,
{ }fuemp => intern_r2emp,
{ }socketcls => intern_r2socketcls,
{ }outsels => intern_r2outsels);,\)

translit(
FOR(BUSNR, 0, NRR2BUSES-1, {    }r2llockreq`'BSUBST\BUSNR <= intern_r2llockreq(BUSNR);
{    }intern_r2squash(BUSNR) <= squash`'BSUBST\BUSNR;
),\)

   fu_internals:mulfui
```

```
{       }port map(
{ }tdata => intern_tdata,
{ }tcls => intern_tcls,
{ }tbsy => intern_tbsy,
{ }toutsels => intern_toutsels,
{ }tdataact => intern_tdataact,

{ }odata => intern_odata,
{ }ocls => intern_ocls,
{ }obsy => intern_obsy,
{ }ooutsels => intern_ooutsels,
{ }odataact => intern_odataact,

{ }r1data => intern_r1data,
{ }r1act => intern_r1act,
{ }r1emp => intern_r1emp,
{ }r1socketcls => intern_r1socketcls,
{ }r1outsels => intern_r1outsels,

{ }r2data => intern_r2data,
{ }r2act => intern_r2act,
{ }r2emp => intern_r2emp,
{ }r2socketcls => intern_r2socketcls,
{ }r2outsels => intern_r2outsels,

{ }ck => ck,
{ }rst => rst);

   valsock:outval
      port map(
outvalid => outvalid,
vin => intern_r1act,
enable => valen);

end func;
------------------------------------------------------------------------------
-- Multiply FU combinational logic
--
--`{ parameter dependencies: MULFUWORDSIZE}'
------------------------------------------------------------------------------

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

entity mulfuc is
   port(a : in std_logic_vector(eval(MULFUWORDSIZE-1) downto 0);
b : in std_logic_vector(eval(MULFUWORDSIZE-1) downto 0);
y1 : out std_logic_vector(eval(MULFUWORDSIZE-1) downto 0);
y2 : out std_logic_vector(eval(MULFUWORDSIZE-1) downto 0);
r1socketcls : in std_logic;
r2socketcls : in std_logic;
socketcls : out std_logic;
r1outsels : in std_logic;
r2outsels : in std_logic;
outsels : out std_logic);
end mulfuc;


architecture func of mulfuc is

begin
   socketcls <= r1socketcls and r2socketcls;
   outsels <= r1outsels or r2outsels;
   process(a,b)
      variable ytemp : std_logic_vector(2*MULFUWORDSIZE-1 downto 0);
   begin
      ytemp := conv_std_logic_vector( conv_integer(signed(a) * signed(b)), ytemp'length);
      y1 <= ytemp(MULFUWORDSIZE-1 downto 0);
      y2 <= ytemp(2*MULFUWORDSIZE-1 downto MULFUWORDSIZE);
   end process;
end func;
------------------------------------------------------------------------------
--  Multiply FU internals
--
--`{ parameter dependencies: BUSWIDTH MULFUWORDSIZE MULFUDELAY}'
------------------------------------------------------------------------------

library IEEE;
use IEEE.Std_Logic_1164.all;

entity mulfui is
   port(
tdata : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
tcls : out std_logic;
tbsy : out std_logic;
toutsels : out std_logic;
```

```
tdataact : in std_logic;

odata : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
ocls : out std_logic;
obsy : out std_logic;
ooutsels : out std_logic;
odataact : in std_logic;

r1data : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
r1act : out std_logic;
r1emp : out std_logic;
r1socketcls : in std_logic;
r1outsels : in std_logic;

r2data : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
r2act : out std_logic;
r2emp : out std_logic;
r2socketcls : in std_logic;
r2outsels : in std_logic;

ck : in std_logic;
rst : in std_logic);
end mulfui;

use work.stagecontrol.all;
use work.combinational.all;

architecture func of mulfui is

   for stagectl0:stagec use entity WORK.stagec(ARCHSTYLE);
   for stagectl1:stagecnb use entity WORK.stagecnb(ARCHSTYLE);
   for comblogic:mulfuc use entity WORK.mulfuc(ARCHSTYLE);

   signal intern_outcls : std_logic;
   signal intern_outactive : std_logic;
   signal intern_outempty : std_logic;
   signal intern_inbsy : std_logic;
   signal intern_incls : std_logic;
   signal intern_ract : std_logic;
   signal intern_remp : std_logic;

   signal stg0_enable : std_logic;
   signal stg1_enable : std_logic;

   signal intern_socketcls : std_logic;
   signal intern_outsels : std_logic;

   signal wired_to_one : std_logic;
   signal wired_to_zero : std_logic;

   signal treg : std_logic_vector(eval(MULFUWORDSIZE-1) downto 0);
   signal oreg : std_logic_vector(eval(MULFUWORDSIZE-1) downto 0);
   signal r1reg : std_logic_vector(eval(MULFUWORDSIZE-1) downto 0);
   signal r2reg : std_logic_vector(eval(MULFUWORDSIZE-1) downto 0);

begin
   stagectl0:stagec
generic map(stagedelay => MULFUDELAY)
port map(inactive  => tdataact,
                 inempty   => wired_to_one,
 inbsy     => intern_inbsy,
 incls     => intern_incls,

 ck        => ck,
 rst       => rst,
 socketcls => intern_socketcls,
 enable    => stg0_enable,

 outactive => intern_outactive,
 outempty  => intern_outempty,
 outbsy    => wired_to_zero,
 outcls    => intern_outcls);
   stagectl1:stagecnb
port map(inactive  => intern_outactive,
                 inempty   => intern_outempty,
 incls     => intern_outcls,

 ck        => ck,
 rst       => rst,
 socketcls => intern_socketcls,
 enable    => stg1_enable,

 outactive => intern_ract,
 outempty  => intern_remp,
 outcls    => wired_to_one);
   comblogic:mulfuc
port map(a => treg,
```

```
 b => oreg,
 y1 => r1reg,
 y2 => r2reg,
 r1socketcls => r1socketcls,
 r2socketcls => r2socketcls,
 socketcls => intern_socketcls,
 r1outsels => r1outsels,
 r2outsels => r2outsels,
 outsels => intern_outsels);

   wired_to_one <= '1';
   wired_to_zero <= '0';

   r1act <= intern_ract;
   r2act <= intern_ract;
   r1emp <= intern_remp;
   r2emp <= intern_remp;
   tbsy <= intern_inbsy;
   obsy <= intern_inbsy;
   tcls <= intern_incls;
   ocls <= intern_incls;
   toutsels <= intern_outsels;
   ooutsels <= intern_outsels;

   process
   begin
      wait on ck until ck='1';
      if (stg1_enable='1') then
 r1data(MULFUWORDSIZE-1 downto 0) <= r1reg;
 r2data(MULFUWORDSIZE-1 downto 0) <= r2reg;
      end if;
      if (stg0_enable='1') then
 treg <= tdata(MULFUWORDSIZE-1 downto 0);
      end if;
      if (odataact='1') then
 oreg <= odata(MULFUWORDSIZE-1 downto 0);
      end if;
   end process;

end func;
------------------------------------------------------------------------------
--  Network Control
--
--`{ parameter dependencies: PARMOVES CMPFUS}'
------------------------------------------------------------------------------

--  The network control unit generates the global lock signal from
--  lock request and squash signals, and generates the exception
--  signal from exception requests. Unlike the SoG version, no
--  squash request signals are used.

library IEEE;
use IEEE.std_logic_1164.all;

entity netcontrol is
   port(glockreq : in std_logic;
cmpglock : in std_logic_vector(CMPFUS-1 downto 0);
FOR(BUSNR,0,PARMOVES-1,
_ifelse(_eval(LLOCKREQS`'BUSNR >= 0), 1, llockreq`'BUSNR : in std_logic_vector(LLOCKREQS`'BUSNR downto 0);)
) squash : in std_logic_vector(PARMOVES-1 downto 0);
glock : out std_logic;
excreq : in std_logic_vector(EXCREQS downto 0);
exc : out std_logic);
end netcontrol;

architecture func of netcontrol is
   signal llockor : std_logic_vector(PARMOVES-1 downto 0);
begin

FOR(BUSNR, 0, PARMOVES-1,
_define(`LLOCKS', LLOCKREQS`'BUSNR)
_ifelse(_eval(LLOCKS >= 0), 1, `
   llockor(BUSNR) <= llockreq`'BUSNR`'(0) _for(CNT, 1, LLOCKS, `
      'or llockreq`'BUSNR`'(CNT));')
_ifelse(_eval(LLOCKS < 0), 1, `   'llockor(BUSNR) <= '0';) )

   glock <= glockreq FOR(CNT, 0, CMPFUS-1, or cmpglock(CNT) )
FOR(CNT, 0, PARMOVES-1, {     }or (llockor(CNT) and not squash(CNT))
)     ;

   exc <= excreq(0) FOR(CNT, 1, EXCREQS,{
  }or excreq(CNT));

end func;
------------------------------------------------------------------------------
-- Output socket for Compare FU.
--
```

```
--`{ parameter dependencies: PARMOVES}'
------------------------------------------------------------------------------
library IEEE;
use IEEE.Std_Logic_1164.all;

entity outcmp is
   port(
rout : out std_logic;
sel : in std_logic_vector(eval(PARMOVES-1) downto 0);
squall : in std_logic;
glockreq : out std_logic;
glock : in std_logic;

rin : in std_logic;
fuact : in std_logic;
fuemp : in std_logic;
socketcls : out std_logic;
outsels : out std_logic);
end outcmp;

architecture func of outcmp is
   signal outselss, sel_or : std_logic;
begin
   sel_or <= sel(0) FOR(BUSNR, 1, PARMOVES-1, or sel(BUSNR) );
   outselss <= not squall and sel_or;
   outsels <= outselss;
   socketcls <= glock or not outselss;
   glockreq <= not fuact and (not fuemp and sel_or);
   rout <= rin;
end func;
FOR(COUNTER, 1, PARMOVES,
{define({SOCKNR},} COUNTER{)
------------------------------------------------------------------------------
-- Output socket for the Immediate FU for SOCKNR buses.
--
--`{ parameter dependencies: BUSWIDTH generic parameters}'
------------------------------------------------------------------------------

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

entity outimm`'SOCKNR is
   generic(
FOR(BUSNR, 0, SOCKNR-2,
{ }lsrcsize`'BUSNR : integer;
{ }lowid`'BUSNR : integer;
){ }lsrcsize`'eval(SOCKNR-1) : integer;
{ }lowid`'eval(SOCKNR-1) : integer);

   port(
FOR(BUSNR, 0, SOCKNR-1,
{ }databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
{ }srcbus`'BUSNR : in std_logic_vector(lsrcsize`'BUSNR-1 downto 0);
)FOR(BUSNR, 0, SOCKNR-1,
{ }data`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
{ }osrcbus`'BUSNR : out std_logic_vector(LOG(eval(IMMEDIATES-1))-1 downto 0);
) glock : in std_logic;
ck : in std_logic);
end outimm`'SOCKNR;


architecture func of outimm`'SOCKNR is
   signal selectl : std_logic_vector(eval(SOCKNR-1) downto 0);
begin
FOR(BUSNR, 0, SOCKNR-1,{
   }osrcbus`'BUSNR <= srcbus`'BUSNR`'(LOG(eval(IMMEDIATES-1))-1 downto 0);)
FOR(BUSNR, 0, SOCKNR-1,{
   }databus`'BUSNR <= data`'BUSNR when (selectl(BUSNR) = '1') else (others => 'Z');)

   process
FOR(BUSNR, 0, SOCKNR-1,
{      }variable id`'BUSNR : std_logic_vector(lsrcsize`'BUSNR-1 downto 0);
)   begin
      wait until ck'event and ck='1';
      if not(glock='1') then
FOR(BUSNR, 0, SOCKNR-1,
{  }id`'BUSNR := std_logic_vector(conv_unsigned(lowid`'BUSNR, id`'BUSNR'length));
{  }if srcbus`'BUSNR`'(lsrcsize`'BUSNR-1 downto LOG(eval(IMMEDIATES-1))) =
          id`'BUSNR`'(lsrcsize`'BUSNR-1 downto LOG(eval(IMMEDIATES-1))) then
    selectl(BUSNR) <= '1';
 else
    selectl(BUSNR) <= '0';
 end if;
)      end if;
   end process;
```

```
end func;
})
FOR(COUNTER, 1, PARMOVES,
{define({SOCKNR},} COUNTER()
------------------------------------------------------------------------------
-- output sockets with opcode, minimal control, for SOCKNR buses
--
--`{ parameter dependencies: BUSWIDTH generic parameters}'
------------------------------------------------------------------------------

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

entity outop`'SOCKNR is
   generic(
FOR(BUSNR, 0, SOCKNR-1,
{ }lsrcsize`'BUSNR : integer;
{ }lowid`'BUSNR : integer;
) logopcodes : integer);

   port(
FOR(BUSNR, 0, SOCKNR-1,
{ }databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
{ }srcbus`'BUSNR : in std_logic_vector(`lsrcsize'BUSNR-1 downto 0);
) glock : in std_logic;
ck : in std_logic;

data : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
opcode : out std_logic_vector(logopcodes-1 downto 0));
end outop`'SOCKNR;

architecture func of outop`'SOCKNR is
   signal selectl : std_logic_vector(eval(SOCKNR-1) downto 0);
FOR(BUSNR, 0, SOCKNR-1,
{   }signal `opcodel'BUSNR : std_logic_vector(logopcodes-1 downto 0);
)begin
FOR(BUSNR, 0, SOCKNR-1,{
   }databus`'BUSNR <= data when (selectl(BUSNR) = '1') else (others => 'Z');)
   opcode <= FOR(BUSNR, 0, SOCKNR-1,{opcodel`'BUSNR when (selectl(BUSNR) = '1') else
     })(others => '-');

   process
FOR(BUSNR, 0, SOCKNR-1,
{      }variable id`'BUSNR : std_logic_vector(lsrcsize`'BUSNR-1 downto 0);
)   begin
      wait until ck'event and ck='1';
      if not(glock = '1') then
FOR(BUSNR, 0, SOCKNR-1,
{  }opcodel`'BUSNR <= srcbus`'BUSNR`(logopcodes-1 downto 0)';
{  }id`'BUSNR := std_logic_vector(conv_unsigned(lowid`'BUSNR, id`'BUSNR'length));
{  }if (srcbus`'BUSNR=id`'BUSNR) then
{     }selectl(BUSNR) <= '1';
{  }else
{     }selectl(BUSNR) <= '0';
{  }end if;
)      end if;
   end process;
end func;


})
FOR(COUNTER, 1, PARMOVES,
{define({SOCKNR},} COUNTER()
------------------------------------------------------------------------------
-- Output socket for register FU, for SOCKNR buses
--
--`{ parameter dependencies: BUSWIDTH generic parameters}'
------------------------------------------------------------------------------

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

entity outreg`'SOCKNR is
   generic(
FOR(BUSNR, 0, SOCKNR-2,
{ }lsrcsize`'BUSNR : integer;
{ }lowid`'BUSNR : integer;
){ }lsrcsize`'eval(SOCKNR-1) : integer;
{ }lowid`'eval(SOCKNR-1) : integer);

   port(
FOR(BUSNR, 0, SOCKNR-1,
{ }databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
{ }srcbus`'BUSNR : in std_logic_vector(lsrcsize`'BUSNR-1 downto 0);
) glock : in std_logic;
```

```
ck : in std_logic;

data : in std_logic_vector(eval(BUSWIDTH-1) downto 0));
end `outreg'SOCKNR;

architecture func of outreg`'SOCKNR is
   signal selectl : std_logic_vector(eval(SOCKNR-1) downto 0);
begin
FOR(BUSNR, 0, SOCKNR-1,{
   }databus`'BUSNR <= data when (selectl(BUSNR) = '1') else (others => 'Z');)

   process
FOR(BUSNR, 0, SOCKNR-1,
{      }variable id`'BUSNR : std_logic_vector(lsrcsize`'BUSNR-1 downto 0);
)   begin
      wait until ck'event and ck='1';
      if not(glock = '1') then
FOR(BUSNR, 0, SOCKNR-1,
{  }id`'BUSNR := std_logic_vector(conv_unsigned(lowid`'BUSNR, id`'BUSNR'length));
{  }if (srcbus`'BUSNR=id`'BUSNR) then
{     }selectl(BUSNR) <= '1';
{  }else
{     }selectl(BUSNR) <= '0';
{  }end if;
)      end if;
   end process;
end func;


})
FOR(COUNTER, 1, PARMOVES,
{define({SOCKNR},} COUNTER{)
------------------------------------------------------------------------------
-- Output socket with selection signal
--
--`{ parameter dependencies: BUSWIDTH gneric parameters}'
------------------------------------------------------------------------------

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

entity outsel`'SOCKNR is
   generic(
FOR(BUSNR, 0, SOCKNR-2,
{ }lsrcsize`'BUSNR : integer;
{ }lowid`'BUSNR : integer;
){ }lsrcsize`'eval(SOCKNR-1) : integer;
{ }lowid`'eval(SOCKNR-1) : integer);

   port(
FOR(BUSNR, 0, SOCKNR-1,
{ }databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
{ }srcbus`'BUSNR : in std_logic_vector(lsrcsize`'BUSNR-1 downto 0);
) squash : in std_logic_vector(eval(SOCKNR-1) downto 0);
glock : in std_logic;
ck : in std_logic;

data : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
       outsels : out std_logic);
end outsel`'SOCKNR;

architecture func of outsel`'SOCKNR is
   signal selectl : std_logic_vector(eval(SOCKNR-1) downto 0);
begin
   outsels <= (not squash(0) and selectl(0))
FOR(BUSNR, 1, SOCKNR-1,
{         }or (not squash(BUSNR) and selectl(BUSNR))
)   ;

FOR(BUSNR, 0, SOCKNR-1,{
   }databus`'BUSNR <= data when (selectl(BUSNR) = '1') else (others => 'Z');)

   process
FOR(BUSNR, 0, SOCKNR-1,
{      }variable id`'BUSNR : std_logic_vector(lsrcsize`'BUSNR-1 downto 0);
)   begin
      wait on ck until ck='1';
      if (glock = '0') then
FOR(BUSNR, 0, SOCKNR-1,
{  }id`'BUSNR := std_logic_vector(conv_unsigned(lowid`'BUSNR, id`'BUSNR'length));
{  }if (srcbus`'BUSNR=id`'BUSNR) then
{     }selectl(BUSNR) <= '1';
{  }else
{     }selectl(BUSNR) <= '0';
{  }end if;
)      end if;
```

```
   end process;
end func;


})
FOR(COUNTER, 1, PARMOVES,
{define({SOCKNR},} COUNTER{)
------------------------------------------------------------------------------
-- Output sockets, no opcode
--
--`{ parameter dependencies: BUSWIDTH generic parameters}'
------------------------------------------------------------------------------

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

entity outsnp`'SOCKNR is
   generic(
FOR(BUSNR, 0, SOCKNR-2,
{ }lsrcsize`'BUSNR : integer;
{ }lowid`'BUSNR : integer;
){ }lsrcsize`'eval(SOCKNR-1) : integer;
{ }lowid`'eval(SOCKNR-1) : integer);

   port(
FOR(BUSNR, 0, SOCKNR-1,
{ }databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
{ }srcbus`'BUSNR : in std_logic_vector(lsrcsize`'BUSNR-1 downto 0);
) excreq : out std_logic;
llockreq : out std_logic_vector(eval(SOCKNR-1) downto 0);
squash : in std_logic_vector(eval(SOCKNR-1) downto 0);
glock : in std_logic;
ck : in std_logic;

data : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
fuact : in std_logic;
fuemp : in std_logic;
socketcls : out std_logic;
outsels : out std_logic);
end outsnp`'SOCKNR;

architecture func of outsnp`'SOCKNR is
   signal selectl : std_logic_vector(eval(SOCKNR-1) downto 0);
   signal outselss : std_logic;
begin
   outselss <= (not squash(0) and selectl(0))
FOR(BUSNR, 1, SOCKNR-1,
{   }or (not squash(BUSNR) and selectl(BUSNR))
)   ;
   outsels <= outselss;
   socketcls <= glock or not outselss;
FOR(BUSNR, 0, SOCKNR-1,
{   }llockreq(BUSNR) <= selectl(BUSNR) and (not fuact and not fuemp);
)   excreq <= fuemp and outselss;

FOR(BUSNR, 0, SOCKNR-1,{
   }databus`'BUSNR <= data when ((not(squash(BUSNR)) and selectl(BUSNR)) = '1') else (others => 'Z');)

   process
FOR(BUSNR, 0, SOCKNR-1,
{      }variable id`'BUSNR : std_logic_vector(lsrcsize`'BUSNR-1 downto 0);
)   begin
      wait until ck'event and ck='1';
      if not(glock = '1') then
FOR(BUSNR, 0, SOCKNR-1,
{  }id`'BUSNR := std_logic_vector(conv_unsigned(lowid`'BUSNR, id`'BUSNR'length));
{  }if (srcbus`'BUSNR=id`'BUSNR) then
{     }selectl(BUSNR) <= '1';
{  }else
{     }selectl(BUSNR) <= '0';
{  }end if;
)      end if;
   end process;
end func;


})
------------------------------------------------------------------------------
-- Output 'socket' for Valid FU
--
--`{ parameter dependencies:}'
------------------------------------------------------------------------------
library IEEE;
use IEEE.Std_Logic_1164.all;

entity outval is
```

```
   port(outvalid : out std_logic;
vin : in std_logic;
enable : in std_logic);
end outval;

architecture func of outval is

begin
process(enable, vin)
   begin
      if (enable = '1') then
 outvalid <= vin;
      else
 outvalid <= 'Z';
      end if;
   end process;
end func;
-------------------------------------------------------------------------------
-- Ifetch FU PC chain control
--
-- parameter dependencies: none
-------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity pccctl is
   port(
pccif_en : out std_logic;
pccdc_en : out std_logic;
pccmv_en : out std_logic;

excmode : out std_logic;

squall : out std_logic;

tdataact : in std_logic;
nopcwrite : in std_logic;
trap : in std_logic;

r2outsels : in std_logic;

glock : in std_logic;

exc : in std_logic;
int : in std_logic;
rst : in std_logic;

ck : in std_logic);
end pccctl;


architecture func of pccctl is
   signal ifen, dcen, mven, squashdc, squashmv, excgens, excmodes : std_logic;
begin
   process
   begin
      wait until ck'event and ck='1';
      mven <= mven or (r2outsels and not glock and dcen); -- set
      dcen <= dcen or (r2outsels and not glock and ifen);
      ifen <= ifen or (r2outsels and not glock);
      if not(glock='1') then
 squashmv <= squashdc;
 squashdc <= '0';
      end if;
      if (excgens='1') then -- overriding reset
 ifen <= '0';
 dcen <= '0';
 mven <= '0';
 squashdc <= '1';
 squashmv <= '1';
      end if;
   end process;

   pccif_en <= (ifen or r2outsels) and not glock;
   pccdc_en <= (dcen or r2outsels) and not glock;
   pccmv_en <= (mven or r2outsels) and not glock;

   excmodes <= not(ifen and dcen and mven);
   excgens <= not rst or (not glock and not excmodes and
(exc or (tdataact and not nopcwrite and trap) or int));
   excmode <= excmodes;
   squall <= squashmv;
end func;
-------------------------------------------------------------------------------
-- Ifetch FU PC incrementer
--
```

```
-- '{parameter dependencies: IFEADDRSIZE}'
-------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity pcincr is
   port(i : in std_logic_vector(IFEADDRSIZE-1 downto 0);
y : out std_logic_vector(IFEADDRSIZE-1 downto 0) );
end pcincr;

architecture func of pcincr is
begin
      y <= conv_std_logic_vector(unsigned(i) + 1, y'length);
end func;
-------------------------------------------------------------------------------
-- Ifetch FU PC selector
--
-- '{parameter dependencies: IFEADDRSIZE IFEOFFSETSIZE IFERSTVEC IFEEXCVEC
--     IFEINTVEC}'
-------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity pcsel is
   port(incpc : in std_logic_vector(IFEADDRSIZE-1 downto 0);
tdata : in std_logic_vector(IFEADDRSIZE-1 downto 0);
tdataact : in std_logic;
offonly : in std_logic;
nopcwrite : in std_logic;
rst : in std_logic;
excmode : in std_logic;
exc : in std_logic;
trap : in std_logic;
int : in std_logic;
newpc : out std_logic_vector(IFEADDRSIZE-1 downto 0);
newpcc : out std_logic_vector(IFEADDRSIZE-1 downto 0));
end pcsel;

architecture func of pcsel is
   signal snewpcc : std_logic_vector(eval(IFEADDRSIZE-1) downto 0);
begin
   snewpcc(eval(IFEOFFSETSIZE-1) downto 0) <= tdata(eval(IFEOFFSETSIZE-1) downto 0)
when ((tdataact and not nopcwrite) ='1') else
      incpc(eval(IFEOFFSETSIZE-1) downto 0);

   snewpcc(eval(IFEADDRSIZE-1) downto IFEOFFSETSIZE) <= tdata(eval(IFEADDRSIZE-1) downto IFEOFFSETSIZE)
when ((tdataact and not nopcwrite and not offonly) = '1') else
      incpc(eval(IFEADDRSIZE-1) downto IFEOFFSETSIZE);
   newpcc <= snewpcc;

   process(rst, excmode, exc, snewpcc, tdataact, nopcwrite, trap, int)
   begin
      if not(rst='1') then
 newpc <= "FOR(CNT,12,IFEADDRSIZE-1,'0')IFERSTVEC";
      elsif (excmode='1') then
    newpc <= snewpcc;
 elsif (exc='1') then
       newpc <= "FOR(CNT,12,IFEADDRSIZE-1,'0')IFEEXCVEC";
    elsif ((tdataact and not nopcwrite and trap) = '1') then
  newpc <= snewpcc;
       elsif (int='1') then
     newpc <= "FOR(CNT,12,IFEADDRSIZE-1,'0')IFEINTVEC";
  else
     newpc <= snewpcc;
      end if;
   end process;

end func;
-------------------------------------------------------------------------------
-- Package with combinational logic entities
--
-- '{parameter dependencies: BUSWIDTH MULFUWORDSIZE}'
-------------------------------------------------------------------------------
library IEEE;
use IEEE.Std_Logic_1164.all;

package combinational is
 component mulfuc
   port(a : in std_logic_vector(eval(MULFUWORDSIZE-1) downto 0);
b : in std_logic_vector(eval(MULFUWORDSIZE-1) downto 0);
y1 : out std_logic_vector(eval(MULFUWORDSIZE-1) downto 0);
y2 : out std_logic_vector(eval(MULFUWORDSIZE-1) downto 0);
r1socketcls : in std_logic;
r2socketcls : in std_logic;
```

```
socketcls : out std_logic;
r1outsels : in std_logic;
r2outsels : in std_logic;
outsels : out std_logic);
 end component;

 component intfuc
   port(
a : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
b : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
opc : in std_logic_vector(LOG(eval(INTFUTS-1))-1 downto 0);
y : out std_logic_vector(eval(BUSWIDTH-1) downto 0));
 end component;

 component cmpfuc
   port(a : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
b : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
opc : in std_logic_vector(LOG(eval(CMPFUTS-1))-1 downto 0);
y : out std_logic);
 end component;

 component logfuc
   port(a : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
b : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
opc : in std_logic_vector(LOG(eval(LOGFUTS-1))-1 downto 0);
y : out std_logic_vector(eval(BUSWIDTH-1) downto 0));
 end component;

 component shifuc
   port(a : in std_logic_vector(BUSWIDTH-1 downto 0);
opc : in std_logic_vector(LOG(eval(SHIFUTS-1))-1 downto 0);
y1 : out std_logic_vector(BUSWIDTH-1 downto 0);
y2 : out std_logic_vector(BUSWIDTH-1 downto 0);
r1socketcls : in std_logic;
r2socketcls : in std_logic;
socketcls : out std_logic;
r1outsels : in std_logic;
r2outsels : in std_logic;
outsels : out std_logic);
 end component;

 component ldsfuc
   port(st0bsy : in std_logic;
wnr : in std_logic;
wnro : out std_logic;
obsy : out std_logic;
st0outactive : in std_logic;
st1inactive : out std_logic;
st0outempty : in std_logic;
st1inempty : out std_logic);
 end component;

end combinational;
-------------------------------------------------------------------------------
-- Functional Units package
--
-- parameter dependencies: PARMOVES BUSWIDTH XXXFUS all connection defines
-- SINGLEGUARD GUARDSPECSIZE
-------------------------------------------------------------------------------

library IEEE;
use IEEE.Std_Logic_1164.all;


package funcunits is

FOR(CNT,0,CMPFUS-1,define({FUNR}, CNT){
 component cmpfu`'FUNR
   port(
FOR(BUSNR, 0, PARMOVES-1,
--
_ifelse(_eval((CCMPFU`'FUNR`'_TBUS`'BUSNR || CCMPFU`'FUNR`'_OBUS`'BUSNR) &&
CCMPFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : inout std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval((CCMPFU`'FUNR`'_TBUS`'BUSNR || CCMPFU`'FUNR`'_OBUS`'BUSNR) &&
!CCMPFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval(!(CCMPFU`'FUNR`'_TBUS`'BUSNR || CCMPFU`'FUNR`'_OBUS`'BUSNR) &&
CCMPFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);)
_ifelse(_eval(CCMPFU`'FUNR`'_TBUS`'BUSNR || CCMPFU`'FUNR`'_OBUS`'BUSNR ||
CCMPFU`'FUNR`'_RBUS`'BUSNR), 1,
squash`'BUSNR : in std_logic;)
_ifelse(_eval(CCMPFU`'FUNR`'_TBUS`'BUSNR || CCMPFU`'FUNR`'_OBUS`'BUSNR), 1,
dstbus`'BUSNR : in std_logic_vector(DSTSIZE`'BUSNR-1 downto 0);)
_ifelse( CCMPFU`'FUNR`'_RBUS`'BUSNR, 1,
srcbus`'BUSNR : in std_logic_vector(SRCSIZE`'BUSNR-1 downto 0);)
```

```
)--
texcreq : out std_logic;
oexcreq : out std_logic;
squall : in std_logic;
glock : in std_logic;
glockreq : out std_logic;
sel : in std_logic_vector(eval(PARMOVES-1) downto 0);
rout : out std_logic;
outvalid : out std_logic;
valen : in std_logic;

ck : in std_logic;
rst : in std_logic
   );
 end component;
})

 component grdfu
   port(
ifdef({SINGLEGUARD},,FOR(BUSNR, 0, PARMOVES-1,
`'guardspec`'BUSNR : in std_logic_vector(eval(GUARDSPECSIZE-1) downto 0);
))
ifdef({SINGLEGUARD},{ }guardspec : in std_logic_vector(eval(GUARDSPECSIZE-1) downto 0);)
squall : in std_logic;
FOR(BUSNR, 0, CMPFUS-1,
{ }cmprout`'BUSNR : in std_logic;
{ }cmpsel`'BUSNR : out std_logic_vector(PARMOVES-1 downto 0);
)
FOR(BUSNR, 0, PARMOVES-1,
{ }squash`'BUSNR : out std_logic;
)
glock : in std_logic;
ck : in std_logic);
 end component;

undefine({FUNR})
define({FUNR}, 0)
 component ifefu`'FUNR
   port(
FOR(BUSNR, 0, PARMOVES-1,
--
_ifelse(_eval(CIFEFU`'FUNR`'_TBUS`'BUSNR && (CIFEFU`'FUNR`'_R1BUS`'BUSNR ||
        CIFEFU`'FUNR`'_R2BUS`'BUSNR)), 1,
        databus`'BUSNR : inout std_logic_vector(eval(BUSWIDTH-1) downto 0);,
        _eval(CIFEFU`'FUNR`'_TBUS`'BUSNR && !(CIFEFU`'FUNR`'_R1BUS`'BUSNR ||
        CIFEFU`'FUNR`'_R2BUS`'BUSNR)), 1,
        databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);,
        _eval(!CIFEFU`'FUNR`'_TBUS`'BUSNR && (CIFEFU`'FUNR`'_R1BUS`'BUSNR ||
        CIFEFU`'FUNR`'_R2BUS`'BUSNR)), 1,
        databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);)
_ifelse(_eval(CIFEFU`'FUNR`'_TBUS`'BUSNR || CIFEFU`'FUNR`'_R2BUS`'BUSNR), 1,
        squash`'BUSNR : in std_logic;)
)
FOR(BUSNR, 0, PARMOVES-1,
{ }srcbus`'BUSNR : out std_logic_vector(_sub(SRCSIZE`'BUSNR)-1 downto 0);
{ }dstbus`'BUSNR : out std_logic_vector(_sub(DSTSIZE`'BUSNR)-1 downto 0);
ifdef({SINGLEGUARD},,`'guardspec`'BUSNR : out std_logic_vector(eval(GUARDSPECSIZE-1) downto 0);)
)
ifdef({SINGLEGUARD},`'guardspec : out std_logic_vector(eval(GUARDSPECSIZE-1) downto 0);)

squall : out std_logic;

iin : in std_logic_vector(eval(INSTRSIZE-1) downto 0);
ia : out std_logic_vector(eval(IFEADDRSIZE-1) downto 0);
ias : out std_logic;
il : out std_logic;

int : in std_logic;

glock : in std_logic;
glockreq : out std_logic;
exc : in std_logic;
ck : in std_logic;
rst : in std_logic);
 end component;

FOR(CNT,0,IMMFUS-1,define({FUNR}, CNT){
 component immfu`'FUNR
   port(
FOR(BUSNR, 0, PARMOVES-1,
--
_ifelse(CIMMFU`'FUNR`'_RBUS`'BUSNR, 1,
databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
srcbus`'BUSNR : in std_logic_vector(SRCSIZE`'BUSNR-1 downto 0);)
)--
glock : in std_logic;
ck : in std_logic);
```

```
 end component;
})

FOR(CNT,0,INTFUS-1,define({FUNR}, CNT){
 component intfu‘’FUNR
   port(
FOR(BUSNR, 0, PARMOVES-1,
--
_ifelse(_eval((CINTFU‘’FUNR‘’_TBUS‘’BUSNR || CINTFU‘’FUNR‘’_OBUS‘’BUSNR) &&
CINTFU‘’FUNR‘’_RBUS‘’BUSNR), 1,
databus‘’BUSNR : inout std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval((CINTFU‘’FUNR‘’_TBUS‘’BUSNR || CINTFU‘’FUNR‘’_OBUS‘’BUSNR) &&
!CINTFU‘’FUNR‘’_RBUS‘’BUSNR), 1,
databus‘’BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval(!(CINTFU‘’FUNR‘’_TBUS‘’BUSNR || CINTFU‘’FUNR‘’_OBUS‘’BUSNR) &&
CINTFU‘’FUNR‘’_RBUS‘’BUSNR), 1,
databus‘’BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);)
_ifelse(_eval(CINTFU‘’FUNR‘’_TBUS‘’BUSNR || CINTFU‘’FUNR‘’_OBUS‘’BUSNR ||
CINTFU‘’FUNR‘’_RBUS‘’BUSNR), 1,
squash‘’BUSNR : in std_logic;)
_ifelse(_eval(CINTFU‘’FUNR‘’_TBUS‘’BUSNR || CINTFU‘’FUNR‘’_OBUS‘’BUSNR), 1,
dstbus‘’BUSNR : in std_logic_vector(DSTSIZE‘’BUSNR-1 downto 0);)
_ifelse( CINTFU‘’FUNR‘’_RBUS‘’BUSNR, 1,
srcbus‘’BUSNR : in std_logic_vector(SRCSIZE‘’BUSNR-1 downto 0);)
_ifelse( CINTFU‘’FUNR‘’_TBUS‘’BUSNR, 1,
tllockreq‘’BUSNR : out std_logic;)
_ifelse( CINTFU‘’FUNR‘’_OBUS‘’BUSNR,
1, ollockreq‘’BUSNR : out std_logic;)
_ifelse( CINTFU‘’FUNR‘’_RBUS‘’BUSNR,
1, rllockreq‘’BUSNR : out std_logic;)
)--
texcreq : out std_logic;
oexcreq : out std_logic;
rexcreq : out std_logic;
glock : in std_logic;
outvalid : out std_logic;
valen : in std_logic;
ck : in std_logic;
rst : in std_logic
   );
 end component;
})

FOR(CNT,0,LDSFUS-1,define({FUNR}, CNT){
 component ldsfu‘’FUNR
   port(
FOR(BUSNR, 0, PARMOVES-1,
--
_ifelse(_eval((CLDSFU‘’FUNR‘’_TBUS‘’BUSNR || CLDSFU‘’FUNR‘’_OBUS‘’BUSNR) &&
CLDSFU‘’FUNR‘’_RBUS‘’BUSNR), 1,
databus‘’BUSNR : inout std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval((CLDSFU‘’FUNR‘’_TBUS‘’BUSNR || CLDSFU‘’FUNR‘’_OBUS‘’BUSNR) &&
!CLDSFU‘’FUNR‘’_RBUS‘’BUSNR), 1,
databus‘’BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval(!(CLDSFU‘’FUNR‘’_TBUS‘’BUSNR || CLDSFU‘’FUNR‘’_OBUS‘’BUSNR) &&
CLDSFU‘’FUNR‘’_RBUS‘’BUSNR), 1,
databus‘’BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);)
_ifelse(_eval(CLDSFU‘’FUNR‘’_TBUS‘’BUSNR || CLDSFU‘’FUNR‘’_OBUS‘’BUSNR ||
CLDSFU‘’FUNR‘’_RBUS‘’BUSNR), 1,
squash‘’BUSNR : in std_logic;)
_ifelse(_eval(CLDSFU‘’FUNR‘’_TBUS‘’BUSNR || CLDSFU‘’FUNR‘’_OBUS‘’BUSNR), 1,
dstbus‘’BUSNR : in std_logic_vector(DSTSIZE‘’BUSNR-1 downto 0);)
_ifelse( CLDSFU‘’FUNR‘’_RBUS‘’BUSNR, 1,
srcbus‘’BUSNR : in std_logic_vector(SRCSIZE‘’BUSNR-1 downto 0);)
_ifelse( CLDSFU‘’FUNR‘’_TBUS‘’BUSNR, 1,
tllockreq‘’BUSNR : out std_logic;)
_ifelse( CLDSFU‘’FUNR‘’_OBUS‘’BUSNR,
1, ollockreq‘’BUSNR : out std_logic;)
_ifelse( CLDSFU‘’FUNR‘’_RBUS‘’BUSNR,
1, rllockreq‘’BUSNR : out std_logic;)
)--
rexcreq : out std_logic;
glock : in std_logic;
outvalid : out std_logic;
valen : in std_logic;

dout : out std_logic_vector(BUSWIDTH-1 downto 0);
din : in std_logic_vector(BUSWIDTH-1 downto 0);
wnr : out std_logic;
da : out std_logic_vector(LDSADDRSIZE-1 downto 0);
das : out std_logic;
dl : out std_logic;

ck : in std_logic;
rst : in std_logic);
 end component;
```

```
})

FOR(CNT,0,LOGFUS-1,define({FUNR}, CNT){
 component logfu`'FUNR
   port(
FOR(BUSNR, 0, PARMOVES-1,
--
_ifelse(_eval((CLOGFU`'FUNR`'_TBUS`'BUSNR || CLOGFU`'FUNR`'_OBUS`'BUSNR) &&
CLOGFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : inout std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval((CLOGFU`'FUNR`'_TBUS`'BUSNR || CLOGFU`'FUNR`'_OBUS`'BUSNR) &&
!CLOGFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval(!(CLOGFU`'FUNR`'_TBUS`'BUSNR || CLOGFU`'FUNR`'_OBUS`'BUSNR) &&
CLOGFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);)
_ifelse(_eval(CLOGFU`'FUNR`'_TBUS`'BUSNR || CLOGFU`'FUNR`'_OBUS`'BUSNR ||
CLOGFU`'FUNR`'_RBUS`'BUSNR), 1,
squash`'BUSNR : in std_logic;)
_ifelse(_eval(CLOGFU`'FUNR`'_TBUS`'BUSNR || CLOGFU`'FUNR`'_OBUS`'BUSNR), 1,
dstbus`'BUSNR : in std_logic_vector(DSTSIZE`'BUSNR-1 downto 0);)
_ifelse( CLOGFU`'FUNR`'_RBUS`'BUSNR, 1,
srcbus`'BUSNR : in std_logic_vector(SRCSIZE`'BUSNR-1 downto 0);)
_ifelse( CLOGFU`'FUNR`'_RBUS`'BUSNR,
1, rllockreq`'BUSNR : out std_logic;)
)--
texcreq : out std_logic;
oexcreq : out std_logic;
rexcreq : out std_logic;
glock : in std_logic;
outvalid : out std_logic;
valen : in std_logic;
ck : in std_logic;
rst : in std_logic
   );
 end component;
})

FOR(CNT,0,MULFUS-1,define({FUNR}, CNT){
 component mulfu`'FUNR
   port(
FOR(BUSNR, 0, PARMOVES-1,
--
_ifelse(_eval((CMULFU`'FUNR`'_TBUS`'BUSNR || CMULFU`'FUNR`'_OBUS`'BUSNR) &&
(CMULFU`'FUNR`'_R1BUS`'BUSNR || CMULFU`'FUNR`'_R2BUS`'BUSNR)), 1,
databus`'BUSNR : inout std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval((CMULFU`'FUNR`'_TBUS`'BUSNR || CMULFU`'FUNR`'_OBUS`'BUSNR) &&
!(CMULFU`'FUNR`'_R1BUS`'BUSNR || CMULFU`'FUNR`'_R2BUS`'BUSNR)), 1,
databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval(!(CMULFU`'FUNR`'_TBUS`'BUSNR || CMULFU`'FUNR`'_OBUS`'BUSNR) &&
(CMULFU`'FUNR`'_R1BUS`'BUSNR || CMULFU`'FUNR`'_R2BUS`'BUSNR)), 1,
databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);)
_ifelse(_eval(CMULFU`'FUNR`'_TBUS`'BUSNR || CMULFU`'FUNR`'_OBUS`'BUSNR ||
CMULFU`'FUNR`'_R1BUS`'BUSNR || CMULFU`'FUNR`'_R2BUS`'BUSNR), 1,
squash`'BUSNR : in std_logic;)
_ifelse(_eval(CMULFU`'FUNR`'_TBUS`'BUSNR || CMULFU`'FUNR`'_OBUS`'BUSNR), 1,
dstbus`'BUSNR : in std_logic_vector(DSTSIZE`'BUSNR-1 downto 0);)
_ifelse(_eval(CMULFU`'FUNR`'_R1BUS`'BUSNR || CMULFU`'FUNR`'_R2BUS`'BUSNR), 1,
srcbus`'BUSNR : in std_logic_vector(SRCSIZE`'BUSNR-1 downto 0);)
_ifelse( CMULFU`'FUNR`'_TBUS`'BUSNR, 1,
tllockreq`'BUSNR : out std_logic;)
_ifelse( CMULFU`'FUNR`'_OBUS`'BUSNR,
1, ollockreq`'BUSNR : out std_logic;)
_ifelse( CMULFU`'FUNR`'_R1BUS`'BUSNR,
1, r1llockreq`'BUSNR : out std_logic;)
_ifelse( CMULFU`'FUNR`'_R2BUS`'BUSNR,
1, r2llockreq`'BUSNR : out std_logic;)
)--
texcreq : out std_logic;
oexcreq : out std_logic;
r1excreq : out std_logic;
r2excreq : out std_logic;
glock : in std_logic;

outvalid : out std_logic;
valen : in std_logic;

ck : in std_logic;
rst : in std_logic);
 end component;
})

FOR(CNT,0,REGFUS-1,define({FUNR}, CNT){
 component regfu`'FUNR
   port(
FOR(BUSNR, 0, PARMOVES-1,
--
```

```
_ifelse(_eval(CREGFU`'FUNR`'_TBUS`'BUSNR && CREGFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : inout std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval(CREGFU`'FUNR`'_TBUS`'BUSNR && !CREGFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval(!CREGFU`'FUNR`'_TBUS`'BUSNR && CREGFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);)
_ifelse(CREGFU`'FUNR`'_TBUS`'BUSNR, 1,
squash`'BUSNR : in std_logic;)
_ifelse(CREGFU`'FUNR`'_TBUS`'BUSNR, 1,
dstbus`'BUSNR : in std_logic_vector(DSTSIZE`'BUSNR-1 downto 0);)
_ifelse(CREGFU`'FUNR`'_RBUS`'BUSNR, 1,
srcbus`'BUSNR : in std_logic_vector(SRCSIZE`'BUSNR-1 downto 0);)
) glock : in std_logic;
ck : in std_logic);
 end component;
})

FOR(CNT,0,RFLFUS-1,define({FUNR}, CNT){
 component rflfu`'FUNR
   port(
FOR(BUSNR, 0, PARMOVES-1,
--
_ifelse(_eval(CRFLFU`'FUNR`'_TBUS`'BUSNR && CRFLFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : inout std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval(CRFLFU`'FUNR`'_TBUS`'BUSNR && !CRFLFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval(!CRFLFU`'FUNR`'_TBUS`'BUSNR && CRFLFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);)
_ifelse(CRFLFU`'FUNR`'_TBUS`'BUSNR, 1,
squash`'BUSNR : in std_logic;)
_ifelse(CRFLFU`'FUNR`'_TBUS`'BUSNR, 1,
dstbus`'BUSNR : in std_logic_vector(DSTSIZE`'BUSNR-1 downto 0);)
_ifelse(CRFLFU`'FUNR`'_RBUS`'BUSNR, 1,
srcbus`'BUSNR : in std_logic_vector(SRCSIZE`'BUSNR-1 downto 0);)
) glock : in std_logic;
ck : in std_logic);
 end component;
})

FOR(CNT,0,SCRFUS-1,define({FUNR}, CNT){
 component scrfu`'FUNR
   port(
FOR(BUSNR, 0, PARMOVES-1,
--
_ifelse(_eval(CINTFU`'FUNR`'_TBUS`'BUSNR && CSCRFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : inout std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval(CSCRFU`'FUNR`'_TBUS`'BUSNR && !CSCRFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval(!CSCRFU`'FUNR`'_TBUS`'BUSNR && CSCRFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);)
_ifelse( CSCRFU`'FUNR`'_TBUS`'BUSNR, 1,
squash`'BUSNR : in std_logic;)
_ifelse( CSCRFU`'FUNR`'_TBUS`'BUSNR, 1,
dstbus`'BUSNR : in std_logic_vector(DSTSIZE`'BUSNR-1 downto 0);)
_ifelse( CSCRFU`'FUNR`'_RBUS`'BUSNR, 1,
srcbus`'BUSNR : in std_logic_vector(SRCSIZE`'BUSNR-1 downto 0);)
)--
glock : in std_logic;

si : in std_logic;
se : in std_logic;

ck : in std_logic;
rst : in std_logic);
 end component;
})

FOR(CNT,0,SHIFUS-1,define({FUNR}, CNT){
 component shifu`'FUNR
   port(
FOR(BUSNR, 0, PARMOVES-1,
--
_ifelse(_eval(CSHIFU`'FUNR`'_TBUS`'BUSNR &&
(CSHIFU`'FUNR`'_R1BUS`'BUSNR || CSHIFU`'FUNR`'_R2BUS`'BUSNR)), 1,
databus`'BUSNR : inout std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval(CSHIFU`'FUNR`'_TBUS`'BUSNR &&
!(CSHIFU`'FUNR`'_R1BUS`'BUSNR || CSHIFU`'FUNR`'_R2BUS`'BUSNR)), 1,
databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval(!CSHIFU`'FUNR`'_TBUS`'BUSNR &&
(CSHIFU`'FUNR`'_R1BUS`'BUSNR || CSHIFU`'FUNR`'_R2BUS`'BUSNR)), 1,
databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);)
_ifelse(_eval(CSHIFU`'FUNR`'_TBUS`'BUSNR ||
CSHIFU`'FUNR`'_R1BUS`'BUSNR || CSHIFU`'FUNR`'_R2BUS`'BUSNR), 1,
squash`'BUSNR : in std_logic;)
_ifelse(CSHIFU`'FUNR`'_TBUS`'BUSNR, 1,
dstbus`'BUSNR : in std_logic_vector(DSTSIZE`'BUSNR-1 downto 0);)
_ifelse(_eval(CSHIFU`'FUNR`'_R1BUS`'BUSNR || CSHIFU`'FUNR`'_R2BUS`'BUSNR), 1,
```

```
srcbus`'BUSNR : in std_logic_vector(SRCSIZE`'BUSNR-1 downto 0);)
_ifelse( CSHIFU`'FUNR`'_R1BUS`'BUSNR,
1, r1llockreq`'BUSNR : out std_logic;)
_ifelse( CSHIFU`'FUNR`'_R2BUS`'BUSNR,
1, r2llockreq`'BUSNR : out std_logic;)
)--
texcreq : out std_logic;
r1excreq : out std_logic;
r2excreq : out std_logic;
glock : in std_logic;
outvalid : out std_logic;
valen : in std_logic;

ck : in std_logic;
rst : in std_logic);
 end component;
})

 component valfu
   port(
FOR(BUSNR, 0, PARMOVES-1,
_ifelse(_eval(CVALFU_RBUS`'BUSNR), 1,
{ }srcbus`'BUSNR : in std_logic_vector(SRCSIZE`'BUSNR-1 downto 0);)
) valen : out std_logic_vector(NVALFU_R-1 downto 0);
glock : in std_logic;
ck : in std_logic);
 end component;

 component netcontrol
   port(glockreq : in std_logic;
cmpglock : in std_logic_vector(CMPFUS-1 downto 0);
FOR(BUSNR,0,PARMOVES-1,
_ifelse(_eval(LLOCKREQS`'BUSNR >= 0), 1, llockreq`'BUSNR : in std_logic_vector(LLOCKREQS`'BUSNR downto 0);)
) squash : in std_logic_vector(PARMOVES-1 downto 0);
glock : out std_logic;
excreq : in std_logic_vector(EXCREQS downto 0);
exc : out std_logic);
 end component;

end funcunits;
-------------------------------------------------------------------------------
-- Control entities for Ifetch FU
--
-- `{parameter dependencies: IFECACHESIZE INSTRSIZE IFECACHERAMASYN }'
-------------------------------------------------------------------------------
library IEEE;
use IEEE.Std_Logic_1164.all;

package ifetchcontrol is

 component icacheram
   port(
cache_addr : in std_logic_vector(eval(LOG(eval(IFECACHESIZE-1))-1) downto 0);
cache_datain : in std_logic_vector(eval(INSTRSIZE-1) downto 0);
cache_tagin : in std_logic_vector(eval(IFETAGSIZE-1) downto 0);

cache_dataout : out std_logic_vector(eval(INSTRSIZE-1) downto 0);
cache_tagout : out std_logic_vector(eval(IFETAGSIZE-1) downto 0);

cache_rwn : in std_logic ifdef({IFECACHERAMASYN},,{;
ck : in std_logic})
);
end component;

 component cchpth
   port(
ifdef({IFECACHERAMASYN},,
{ newifca : in std_logic_vector(LOG(eval(IFECACHESIZE-1))-1 downto 0);
}) ifa : in std_logic_vector(IFEADDRSIZE-1 downto 0);
dca : in std_logic_vector(IFEADDRSIZE-1 downto 0);
cwrite : in std_logic;
rama : out std_logic_vector(LOG(eval(IFECACHESIZE-1))-1 downto 0);
dt : out std_logic_vector(IFETAGSIZE-1 downto 0);
qt : in std_logic_vector(IFETAGSIZE-1 downto 0);
cenabled : in std_logic;
insenforce : in std_logic;
insen : in std_logic;
insvalid : out std_logic;

iin : in std_logic_vector(INSTRSIZE-1 downto 0);
inscache : in std_logic_vector(INSTRSIZE-1 downto 0);
insreg : out std_logic_vector(INSTRSIZE-1 downto 0);

insext : in std_logic;

ifdef({IFECACHERAMASYN},,
{ glock : in std_logic;
```

```
}) rst : in std_logic;
ck : in std_logic);
 end component;

 component cchfsm
   port(
insvalid : in std_logic;
rst : in std_logic;
rwn : out std_logic;
cwrite : out std_logic;
read : out std_logic;
il : out std_logic;
insext : out std_logic;
insenforce : out std_logic;
ck : in std_logic);
 end component;

 component cchctl
   port(
insvalid : in std_logic;
rwn : out std_logic;
cwrite : out std_logic;
ias : out std_logic;
il : out std_logic;
insext : out std_logic;
insen : out std_logic;
insenforce : out std_logic;
glockreq : out std_logic;
glock : in std_logic;
rst : in std_logic;
ck : in std_logic);
 end component;

 component pccctl
   port(
pccif_en : out std_logic;
pccdc_en : out std_logic;
pccmv_en : out std_logic;

excmode : out std_logic;

squall : out std_logic;

tdataact : in std_logic;
nopcwrite : in std_logic;
trap : in std_logic;

r2outsels : in std_logic;

glock : in std_logic;

exc : in std_logic;
int : in std_logic;
rst : in std_logic;

ck : in std_logic);
 end component;

 component pcincr
   port(i : in std_logic_vector(IFEADDRSIZE-1 downto 0);
y : out std_logic_vector(IFEADDRSIZE-1 downto 0) );
 end component;

 component pcsel
   port(incpc : in std_logic_vector(IFEADDRSIZE-1 downto 0);
tdata : in std_logic_vector(IFEADDRSIZE-1 downto 0);
tdataact : in std_logic;
offonly : in std_logic;
nopcwrite : in std_logic;
rst : in std_logic;
excmode : in std_logic;
exc : in std_logic;
trap : in std_logic;
int : in std_logic;
newpc : out std_logic_vector(IFEADDRSIZE-1 downto 0);
newpcc : out std_logic_vector(IFEADDRSIZE-1 downto 0));
 end component;

 component regctl
   port(glock : in std_logic;
pcen : out std_logic;
tdataact : in std_logic;
nopcwrite : in std_logic;
noctlsel : in std_logic;
ctlreg_en : out std_logic;
rst : in std_logic);
 end component;
```

```
end ifetchcontrol;
-----------------------------------------------------------------------------
-- Package with FU internals
--
-- '{parameter dependencies: BUSWIDTH }'
-----------------------------------------------------------------------------
library IEEE;
use IEEE.Std_Logic_1164.all;

package internals is
   component mulfui
   port(
tdata : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
tcls : out std_logic;
tbsy : out std_logic;
toutsels : out std_logic;
tdataact : in std_logic;

odata : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
ocls : out std_logic;
obsy : out std_logic;
ooutsels : out std_logic;
odataact : in std_logic;

r1data : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
r1act : out std_logic;
r1emp : out std_logic;
r1socketcls : in std_logic;
r1outsels : in std_logic;

r2data : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
r2act : out std_logic;
r2emp : out std_logic;
r2socketcls : in std_logic;
r2outsels : in std_logic;

ck : in std_logic;
rst : in std_logic);
   end component;

   component intfui
   port(
tdata : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
topcode : in std_logic_vector(LOG(eval(INTFUTS-1))-1 downto 0);
tcls : out std_logic;
tbsy : out std_logic;
toutsels : out std_logic;
tdataact : in std_logic;

odata : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
ocls : out std_logic;
obsy : out std_logic;
ooutsels : out std_logic;
odataact : in std_logic;

rdata : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
ract : out std_logic;
remp : out std_logic;
rsocketcls : in std_logic;
routsels : in std_logic;

ck : in std_logic;
rst : in std_logic);
   end component;

 component cmpfui
   port(
tdata : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
topcode : in std_logic_vector(LOG(eval(CMPFUTS-1))-1 downto 0);
tcls : out std_logic;
toutsels : out std_logic;
tdataact : in std_logic;

odata : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
ocls : out std_logic;
ooutsels : out std_logic;
odataact : in std_logic;

rdata : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
ract : out std_logic;
remp : out std_logic;
rsocketcls : in std_logic;
routsels : in std_logic;

rout : out std_logic;
```

```
ck : in std_logic;
rst : in std_logic);
 end component;


 component ifefui
   port(
tdata : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
topcode : in std_logic_vector(LOG(eval(IFEFUTS-1))-1 downto 0);
tdataact : in std_logic;

r1data : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
r2data : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
r2outsels : in std_logic;
translit(
FOR(NDX, 0, PARMOVES-1,
{ }srcbus‘’NDX : out std_logic_vector(SRCSIZE\NDX-1 downto 0);
{ }dstbus‘’NDX : out std_logic_vector(DSTSIZE\NDX-1 downto 0);
ifdef({SINGLEGUARD},,{ }guardspec‘’NDX : out std_logic_vector(eval(GUARDSPECSIZE-1) downto 0);)
),\)
ifdef({SINGLEGUARD},{ }guardspec : out std_logic_vector(eval(GUARDSPECSIZE-1) downto 0);)
{ }squall : out std_logic;
{ }iin : in std_logic_vector(eval(INSTRSIZE-1) downto 0);
{ }ia : out std_logic_vector(eval(IFEADDRSIZE-1) downto 0);
{ }ias : out std_logic;
{ }il : out std_logic;

{ }int : in std_logic;
{ }glock : in std_logic;
{ }glockreq : out std_logic;
{ }exc : in std_logic;
{ }ck : in std_logic;
{ }rst : in std_logic);
 end component;


 component logfui
   port(
tdata : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
topcode : in std_logic_vector(LOG(eval(LOGFUTS-1))-1 downto 0);
tcls : out std_logic;
toutsels : out std_logic;
tdataact : in std_logic;

odata : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
ocls : out std_logic;
ooutsels : out std_logic;
odataact : in std_logic;

rdata : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
ract : out std_logic;
remp : out std_logic;
rsocketcls : in std_logic;
routsels : in std_logic;

ck : in std_logic;
rst : in std_logic);
 end component;

 component shifui
   port(
tdata : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
topcode : in std_logic_vector(LOG(eval(SHIFUTS-1))-1 downto 0);
tcls : out std_logic;
toutsels : out std_logic;
tdataact : in std_logic;

r1data : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
r1act : out std_logic;
r1emp : out std_logic;
r1socketcls : in std_logic;
r1outsels : in std_logic;

r2data : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
r2act : out std_logic;
r2emp : out std_logic;
r2socketcls : in std_logic;
r2outsels : in std_logic;

ck : in std_logic;
rst : in std_logic);
 end component;

 component ldsfui
   port(tdata : in std_logic_vector(BUSWIDTH-1 downto 0);
        topcode : in std_logic_vector(LOG(eval(LDSFUTS-1))-1 downto 0);
        tcls : out std_logic;
```

```
        tbsy : out std_logic;
        toutsels : out std_logic;
        tdataact : in std_logic;

        odata : in std_logic_vector(BUSWIDTH-1 downto 0);
        ocls : out std_logic;
        obsy : out std_logic;
        ooutsels : out std_logic;
        odataact : in std_logic;

        rdata : out std_logic_vector(BUSWIDTH-1 downto 0);
        ract : out std_logic;
        remp : out std_logic;
        rsocketcls : in std_logic;
        routsels : in std_logic;

dout : out std_logic_vector(BUSWIDTH-1 downto 0);
din : in std_logic_vector(BUSWIDTH-1 downto 0);
wnr : out std_logic;
da : out std_logic_vector(LDSADDRSIZE-1 downto 0);
das : out std_logic;
dl : out std_logic;

        ck : in std_logic;
        rst : in std_logic);
 end component;

FOR(BUSNR,1,PARMOVES,
{define({FUINR},} BUSNR{)
 component immfui`'FUINR
   port(
FOR(CNT, 0, FUINR-1,{ }rdata`'CNT : out std_logic_vector(BUSWIDTH-1 downto 0);
{ }srcbus`'CNT : in std_logic_vector(LOG(eval(IMMEDIATES-1))-1 downto 0);
) glock : in std_logic;
ck : in std_logic);
 end component;
})

 component regfui
   port(
tdata : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
tdataact : in std_logic;
rdata : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
ck : in std_logic);
 end component;

FOR(CNT,0,RFLFUS-1,
{define({FUINR},} CNT{)
 component rflfui`'FUINR
   port(tdata : in std_logic_vector(BUSWIDTH-1 downto 0);
topcode : in std_logic_vector(_log(_eval(PRFLFU`'FUINR`'_REGS-1))-1 downto 0);
tdataact : in std_logic;
rdata : out std_logic_vector(BUSWIDTH-1 downto 0);
ropcode : in std_logic_vector(_log(_eval(PRFLFU`'FUINR`'_REGS-1))-1 downto 0);
glock : in std_logic;
ck : in std_logic);
 end component;
})

end internals;
-------------------------------------------------------------------------------
-- Package with input/output sockets
--
-- `{parameter dependencies: PARMOVES BUSWIDTH IMMEDIATES}'
-------------------------------------------------------------------------------
library IEEE;
use IEEE.Std_Logic_1164.all;

package sockets is

FOR(SOCKCOUNTER, 1, PARMOVES,
{define({SOCKNR},} SOCKCOUNTER{)
 component insock`'SOCKNR -- normal input sockets, with opcode.
   generic(
FOR(BUSNR, 0, SOCKNR-1,
{ }ldstsize`'BUSNR : integer;
{ }lowid`'BUSNR : integer;
){ }logopcodes : integer);

   port(
FOR(BUSNR, 0, SOCKNR-1,
{ }databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
{ }dstbus`'BUSNR : in std_logic_vector(ldstsize`'BUSNR-1 downto 0);
) excreq : out std_logic;
llockreq : out std_logic_vector(eval(SOCKNR-1) downto 0);
squash : in std_logic_vector(eval(SOCKNR-1) downto 0);
glock : in std_logic;
```

```
ck : in std_logic;

data : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
opcode : out std_logic_vector(logopcodes-1 downto 0);
fucls : in std_logic;
fubsy : in std_logic;
outsels : in std_logic;
dataact : out std_logic);
 end component;
})

FOR(SOCKCOUNTER, 1, PARMOVES,
{define({SOCKNR},} SOCKCOUNTER{)
 component insnop`'SOCKNR  -- input sockets without opcode.
   generic(
FOR(BUSNR, 0, SOCKNR-2,
{ }ldstsize`'BUSNR : integer;
{ }lowid`'BUSNR : integer;
){ }ldstsize`'eval(SOCKNR-1) : integer;
{ }lowid`'eval(SOCKNR-1) : integer);

   port(
FOR(BUSNR, 0, SOCKNR-1,
{ }databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
{ }dstbus`'BUSNR : in std_logic_vector(ldstsize`'BUSNR-1 downto 0);
) excreq : out std_logic;
llockreq : out std_logic_vector(eval(SOCKNR-1) downto 0);
squash : in std_logic_vector(eval(SOCKNR-1) downto 0);
glock : in std_logic;
ck : in std_logic;

data : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
fucls : in std_logic;
fubsy : in std_logic;
outsels : in std_logic;
dataact : out std_logic);
 end component;
})

FOR(SOCKCOUNTER, 1, PARMOVES,
{define({SOCKNR},} SOCKCOUNTER{)

 component insknb`'SOCKNR -- input socket, no bsy
   generic(
FOR(BUSNR, 0, SOCKNR-1,
{ }ldstsize`'BUSNR : integer;
{ }lowid`'BUSNR : integer;
){ }logopcodes : integer);

   port(
FOR(BUSNR, 0, SOCKNR-1,
{ }databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
{ }dstbus`'BUSNR : in std_logic_vector(ldstsize`'BUSNR-1 downto 0);
) excreq : out std_logic;
squash : in std_logic_vector(eval(SOCKNR-1) downto 0);
glock : in std_logic;
ck : in std_logic;

data : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
opcode : out std_logic_vector(logopcodes-1 downto 0);
fucls : in std_logic;
outsels : in std_logic;
dataact : out std_logic);
 end component;
})

FOR(SOCKCOUNTER, 1, PARMOVES,
{define({SOCKNR},} SOCKCOUNTER{)

 component inspnb`'SOCKNR -- input socket, no opcode, no bsy
   generic(
FOR(BUSNR, 0, SOCKNR-2,
{ }ldstsize`'BUSNR : integer := 7;
{ }lowid`'BUSNR : integer := 83;
){ }ldstsize`'eval(SOCKNR-1) : integer := 7;
{ }lowid`'eval(SOCKNR-1) : integer := 85);

   port(
FOR(BUSNR, 0, SOCKNR-1,
{ }databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
{ }dstbus`'BUSNR : in std_logic_vector(ldstsize`'BUSNR-1 downto 0);
) excreq : out std_logic;
squash : in std_logic_vector(eval(SOCKNR-1) downto 0);
glock : in std_logic;
ck : in std_logic;

data : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
```

```
fucls : in std_logic;
outsels : in std_logic;
dataact : out std_logic);
 end component;
})

FOR(SOCKCOUNTER, 1, PARMOVES,
{define({SOCKNR},} SOCKCOUNTER{)

 component inskne`'SOCKNR -- input socket, no exception or bsy support
   generic(
FOR(BUSNR, 0, SOCKNR-1,
{ }ldstsize`'BUSNR : integer;
{ }lowid`'BUSNR : integer;
){ }logopcodes : integer);

   port(
FOR(BUSNR, 0, SOCKNR-1,
{ }databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
{ }dstbus`'BUSNR : in std_logic_vector(ldstsize`'BUSNR-1 downto 0);
) squash : in std_logic_vector(eval(SOCKNR-1) downto 0);
glock : in std_logic;
ck : in std_logic;

data : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
opcode : out std_logic_vector(logopcodes-1 downto 0);
dataact : out std_logic);
 end component;
})

FOR(SOCKCOUNTER, 1, PARMOVES,
{define({SOCKNR},} SOCKCOUNTER{)

 component insreg`'SOCKNR -- input socket for register units
   generic(
FOR(BUSNR, 0, SOCKNR-2,
{ }ldstsize`'BUSNR : integer := 7;
{ }lowid`'BUSNR : integer := 83;
){ }ldstsize`'eval(SOCKNR-1) : integer := 7;
{ }lowid`'eval(SOCKNR-1) : integer := 85);

   port(
FOR(BUSNR, 0, SOCKNR-1,
{ }databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
{ }dstbus`'BUSNR : in std_logic_vector(ldstsize`'BUSNR-1 downto 0);
) squash : in std_logic_vector(eval(SOCKNR-1) downto 0);
glock : in std_logic;
ck : in std_logic;

data : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
dataact : out std_logic);
 end component;
})

FOR(SOCKCOUNTER, 1, PARMOVES,
{define({SOCKNR},} SOCKCOUNTER{)

 component outsnp`'SOCKNR -- output socket without opcode
   generic(
FOR(BUSNR, 0, SOCKNR-2,
{ }lsrcsize`'BUSNR : integer;
{ }lowid`'BUSNR : integer;
){ }lsrcsize`'eval(SOCKNR-1) : integer;
{ }lowid`'eval(SOCKNR-1) : integer);

   port(
FOR(BUSNR, 0, SOCKNR-1,
{ }databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
{ }srcbus`'BUSNR : in std_logic_vector(lsrcsize`'BUSNR-1 downto 0);
) excreq : out std_logic;
llockreq : out std_logic_vector(eval(SOCKNR-1) downto 0);
squash : in std_logic_vector(eval(SOCKNR-1) downto 0);
glock : in std_logic;
ck : in std_logic;

data : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
fuact : in std_logic;
fuemp : in std_logic;
socketcls : out std_logic;
outsels : out std_logic);
 end component;
})

FOR(SOCKCOUNTER, 1, PARMOVES,
{define({SOCKNR},} SOCKCOUNTER{)

 component outreg`'SOCKNR -- output socket for register units
```

```
   generic(
FOR(BUSNR, 0, SOCKNR-2,
{ }lsrcsize`'BUSNR : integer;
{ }lowid`'BUSNR : integer;
){ }lsrcsize`'eval(SOCKNR-1) : integer;
{ }lowid`'eval(SOCKNR-1) : integer);

   port(
FOR(BUSNR, 0, SOCKNR-1,
{ }databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
{ }srcbus`'BUSNR : in std_logic_vector(lsrcsize`'BUSNR-1 downto 0);
) glock : in std_logic;
ck : in std_logic;

data : in std_logic_vector(eval(BUSWIDTH-1) downto 0));
 end component;
})

FOR(SOCKCOUNTER, 1, PARMOVES,
{define({SOCKNR},} SOCKCOUNTER{)

 component outsel`'SOCKNR -- output socket with selection signal
   generic(
FOR(BUSNR, 0, SOCKNR-2,
{ }lsrcsize`'BUSNR : integer;
{ }lowid`'BUSNR : integer;
){ }lsrcsize`'eval(SOCKNR-1) : integer;
{ }lowid`'eval(SOCKNR-1) : integer);

   port(
FOR(BUSNR, 0, SOCKNR-1,
{ }databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
{ }srcbus`'BUSNR : in std_logic_vector(lsrcsize`'BUSNR-1 downto 0);
) squash : in std_logic_vector(eval(SOCKNR-1) downto 0);
glock : in std_logic;
ck : in std_logic;

data : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
        outsels : out std_logic);
 end component;
})

FOR(SOCKCOUNTER, 1, PARMOVES,
{define({SOCKNR},} SOCKCOUNTER{)

 component outimm`'SOCKNR -- output sockets for immediate unit
   generic(
FOR(BUSNR, 0, SOCKNR-2,
{ }lsrcsize`'BUSNR : integer;
{ }lowid`'BUSNR : integer;
){ }lsrcsize`'eval(SOCKNR-1) : integer;
{ }lowid`'eval(SOCKNR-1) : integer);

   port(
FOR(BUSNR, 0, SOCKNR-1,
{ }databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
{ }srcbus`'BUSNR : in std_logic_vector(lsrcsize`'BUSNR-1 downto 0);
)FOR(BUSNR, 0, SOCKNR-1,
{ }data`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
{ }osrcbus`'BUSNR : out std_logic_vector(LOG(eval(IMMEDIATES-1))-1 downto 0);
) glock : in std_logic;
ck : in std_logic);
 end component;
})

 component outcmp -- output socket for compare unit
   port(rout : out std_logic;
sel : in std_logic_vector(eval(PARMOVES-1) downto 0);
squall : in std_logic;
glockreq : out std_logic;
glock : in std_logic;

rin : in std_logic;
fuact : in std_logic;
fuemp : in std_logic;
socketcls : out std_logic;
outsels : out std_logic);
 end component;

 component outval -- output socket for valid unit
   port(outvalid : out std_logic;
vin : in std_logic;
enable : in std_logic);
 end component;

FOR(SOCKCOUNTER, 1, PARMOVES,
{define({SOCKNR},} SOCKCOUNTER{)
```

```
 component outop‘’SOCKNR
   generic(
FOR(BUSNR, 0, SOCKNR-1,
{ }lsrcsize‘’BUSNR : integer;
{ }lowid‘’BUSNR : integer;
) logopcodes : integer);

   port(
FOR(BUSNR, 0, SOCKNR-1,
{ }databus‘’BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
{ }srcbus‘’BUSNR : in std_logic_vector(‘lsrcsize’BUSNR-1 downto 0);
) glock : in std_logic;
ck : in std_logic;

data : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
opcode : out std_logic_vector(logopcodes-1 downto 0));
 end component;
})

end sockets;
-------------------------------------------------------------------------------
-- Package with stagecontrol entities
--
-- parameter dependencies: none
-------------------------------------------------------------------------------
library IEEE;
use IEEE.Std_Logic_1164.all;

package stagecontrol is

   component stagec
      generic(stagedelay : integer);
      port(inactive : in std_logic;
   inempty : in std_logic;
   inbsy : out std_logic;
   incls : out std_logic;

   ck : in std_logic;
   rst : in std_logic;
   socketcls : in std_logic;
   enable : out std_logic;

   outactive : out std_logic;
   outempty : out std_logic;
   outbsy : in std_logic;
   outcls : in std_logic);
   end component;

   component stagecnb
      port(inactive : in std_logic;
   inempty : in std_logic;
   incls : out std_logic;

   ck : in std_logic;
   rst : in std_logic;
   socketcls : in std_logic;
   enable : out std_logic;

   outactive : out std_logic;
   outempty : out std_logic;
   outcls : in std_logic);
   end component;

end stagecontrol;
-------------------------------------------------------------------------------
-- Ifetch FU register control
--
-- parameter dependencies: none
-------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity regctl is
   port(glock : in std_logic;
pcen : out std_logic;
tdataact : in std_logic;
nopcwrite : in std_logic;
noctlsel : in std_logic;
ctlreg_en : out std_logic;
rst : in std_logic);
end regctl;

architecture func of regctl is
begin
   pcen <= not glock or not rst;
```

```
   ctlreg_en <= not glock and tdataact and nopcwrite and not noctlsel;
end func;
-----------------------------------------------------------------------------
-- Register FU
--
-- ‘{parameter dependencies: PARMOVES BUSWIDTH}’
-----------------------------------------------------------------------------
library IEEE;
use IEEE.Std_Logic_1164.all;

entity regfu‘’FUNR is
   port(
FOR(BUSNR, 0, PARMOVES-1,
--
_ifelse(_eval(CREGFU‘’FUNR‘’_TBUS‘’BUSNR && CREGFU‘’FUNR‘’_RBUS‘’BUSNR), 1,
databus‘’BUSNR : inout std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval(CREGFU‘’FUNR‘’_TBUS‘’BUSNR && !CREGFU‘’FUNR‘’_RBUS‘’BUSNR), 1,
databus‘’BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval(!CREGFU‘’FUNR‘’_TBUS‘’BUSNR && CREGFU‘’FUNR‘’_RBUS‘’BUSNR), 1,
databus‘’BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);)
_ifelse(CREGFU‘’FUNR‘’_TBUS‘’BUSNR, 1,
squash‘’BUSNR : in std_logic;)
_ifelse(CREGFU‘’FUNR‘’_TBUS‘’BUSNR, 1,
dstbus‘’BUSNR : in std_logic_vector(DSTSIZE‘’BUSNR-1 downto 0);)
_ifelse(CREGFU‘’FUNR‘’_RBUS‘’BUSNR, 1,
srcbus‘’BUSNR : in std_logic_vector(SRCSIZE‘’BUSNR-1 downto 0);)
) glock : in std_logic;
ck : in std_logic);
end regfu‘’FUNR;

use work.sockets.all;
use work.internals.all;

architecture func of regfu‘’FUNR is
define({NRTBUSES}, translit(NREGFU\FUNR\_T,\))
define({NRRBUSES}, translit(NREGFU\FUNR\_R,\))

   for tsock:insreg‘’NRTBUSES use entity WORK.insreg‘’NRTBUSES‘’(ARCHSTYLE);
   for rsock:outreg‘’NRRBUSES use entity WORK.outreg‘’NRRBUSES‘’(ARCHSTYLE);
   for fu_internals:regfui use entity WORK.regfui(ARCHSTYLE);

   signal intern_tdata : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_tdataact : std_logic;
   signal intern_tsquash : std_logic_vector(NRTBUSES-1 downto 0);
   signal intern_rdata : std_logic_vector(eval(BUSWIDTH-1) downto 0);

begin
   tsock:insreg‘’NRTBUSES
define({BSUBST}, translit(BREGFU\FUNR\_T,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NRTBUSES-2,
{ }ldstsize‘’BUSNR => _sub(DSTSIZE‘’BSUBST\BUSNR){,}
{ }lowid‘’BUSNR => _sub(AREGFU\FUNR\_TBUS‘’BSUBST\BUSNR){,}
){ }ldstsize‘’eval(NRTBUSES-1) =>_sub(DSTSIZE‘’BSUBST\eval(NRTBUSES-1)){,}
{ }lowid‘’eval(NRTBUSES-1) => _sub(AREGFU\FUNR\_TBUS‘’BSUBST\eval(NRTBUSES-1))),\)
translit(
{      }port map(
FOR(BUSNR, 0, NRTBUSES-1,
{ }databus‘’BUSNR => databus‘’BSUBST\BUSNR{,}
{ }dstbus‘’BUSNR => dstbus‘’BSUBST\BUSNR{,}
){ }squash => intern_tsquash,
{ }glock => glock,
{ }ck => ck,
{ }data => intern_tdata,
{ }dataact => intern_tdataact );,\)

translit(
FOR(BUSNR, 0, NRTBUSES-1, {   }intern_tsquash(BUSNR) <= squash‘’BSUBST\BUSNR;
),\)

   rsock:outreg‘’NRRBUSES
define({BSUBST}, translit(BREGFU\FUNR\_R,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NRRBUSES-2,
{ }lsrcsize‘’BUSNR => _sub(SRCSIZE‘’BSUBST\BUSNR){,}
{ }lowid‘’BUSNR => _sub(AREGFU\FUNR\_RBUS‘’BSUBST\BUSNR){,}
){ }lsrcsize‘’eval(NRRBUSES-1) =>_sub(SRCSIZE‘’BSUBST\eval(NRRBUSES-1)){,}
{ }lowid‘’eval(NRRBUSES-1) => _sub(AREGFU\FUNR\_RBUS‘’BSUBST\eval(NRRBUSES-1))),\)
translit(
{      }port map(
FOR(BUSNR, 0, NRRBUSES-1,
{ }databus‘’BUSNR => databus‘’BSUBST\BUSNR{,}
{ }srcbus‘’BUSNR => srcbus‘’BSUBST\BUSNR{,}
){ }glock => glock,
{ }ck => ck,
```

```
{ }data => intern_rdata);,\)

   fu_internals:regfui
      port map(
tdata => intern_tdata,
tdataact => intern_tdataact,
rdata => intern_rdata,
ck => ck);

end func;
------------------------------------------------------------------------------
-- Register FU internals
--
-- `{parameter dependencies: BUSWIDTH}'
------------------------------------------------------------------------------

library IEEE;
use IEEE.Std_Logic_1164.all;

entity regfui is
   port(
tdata : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
tdataact : in std_logic;
rdata : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
ck : in std_logic);
end regfui;


architecture func of regfui is
   signal treg : std_logic_vector(eval(BUSWIDTH-1) downto 0);
begin
   process
   begin
      wait until ck'event and ck='1';
      if (tdataact = '1') then
 treg <= tdata;
      end if;
   end process;
   rdata <= treg;
end func;
------------------------------------------------------------------------------
-- Register file FU
--
-- `{parameter dependencies: PARMOVES BUSWIDTH PRFLFU._REGS connection defines}'
------------------------------------------------------------------------------
library IEEE;
use IEEE.Std_Logic_1164.all;
use work.sockets.all;
use work.internals.all;


entity rflfu`'FUNR is
   port(
FOR(BUSNR, 0, PARMOVES-1,
--
_ifelse(_eval(CRFLFU`'FUNR`'_TBUS`'BUSNR && CRFLFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : inout std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval(CRFLFU`'FUNR`'_TBUS`'BUSNR && !CRFLFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval(!CRFLFU`'FUNR`'_TBUS`'BUSNR && CRFLFU`'FUNR`'_RBUS`'BUSNR), 1,
databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);)
_ifelse(CRFLFU`'FUNR`'_TBUS`'BUSNR, 1,
squash`'BUSNR : in std_logic;)
_ifelse(CRFLFU`'FUNR`'_TBUS`'BUSNR, 1,
dstbus`'BUSNR : in std_logic_vector(DSTSIZE`'BUSNR-1 downto 0);)
_ifelse(CRFLFU`'FUNR`'_RBUS`'BUSNR, 1,
srcbus`'BUSNR : in std_logic_vector(SRCSIZE`'BUSNR-1 downto 0);)
) glock : in std_logic;
ck : in std_logic);
end rflfu`'FUNR;


architecture func of rflfu`'FUNR is
define({NRTBUSES}, translit(NRFLFU\FUNR\_T,\))
define({NRRBUSES}, translit(NRFLFU\FUNR\_R,\))
   for tsock:inskne`'NRTBUSES use entity work.inskne`'NRTBUSES`'(ARCHSTYLE);
   for rsock:outop`'NRRBUSES use entity work.outop`'NRRBUSES`'(ARCHSTYLE);
   for internals:rflfui`'FUNR use entity work.rflfui`'FUNR`'(ARCHSTYLE);

   signal intern_tdata : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_topcode : std_logic_vector(_log(_eval(PRFLFU`'FUNR`'_REGS-1))-1 downto 0);
   signal intern_tdataact : std_logic;
   signal intern_tsquash : std_logic_vector(NRTBUSES-1 downto 0);

   signal intern_rdata : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_ropcode : std_logic_vector(_log(_eval(PRFLFU`'FUNR`'_REGS-1))-1 downto 0);
```

```
begin
   tsock:inskne‘’NRTBUSES
define({BSUBST}, translit(BRFLFU\FUNR\_T,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NRTBUSES-1,
{ }ldstsize‘’BUSNR => _sub(DSTSIZE‘’BSUBST\BUSNR){,}
{ }lowid‘’BUSNR => _sub(ARFLFU\FUNR\_TBUS‘’BSUBST\BUSNR){,}
){ }logopcodes => _log(_eval(PRFLFU‘’FUNR‘’_REGS-1))) ,\)
translit(
{      }port map(
FOR(BUSNR, 0, NRTBUSES-1,
{ }databus‘’BUSNR => databus‘’BSUBST\BUSNR{,}
{ }dstbus‘’BUSNR => dstbus‘’BSUBST\BUSNR{,}
){ }squash => intern_tsquash{,}
{ }glock => glock{,}
{ }ck => ck,
{ }data => intern_tdata,
{ }opcode => intern_topcode,
{ }dataact => intern_tdataact );
,\)
translit(
FOR(BUSNR, 0, NRTBUSES-1, {   }intern_tsquash(BUSNR) <= squash‘’BSUBST\BUSNR;
),\)

   rsock:outop‘’NRRBUSES
define({BSUBST}, translit(BRFLFU\FUNR\_T,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NRRBUSES-1,
{ }lsrcsize‘’BUSNR => _sub(SRCSIZE‘’BSUBST\BUSNR){,}
{ }lowid‘’BUSNR => _sub(ARFLFU\FUNR\_RBUS‘’BSUBST\BUSNR){,}
){ }logopcodes => _log(_eval(PRFLFU‘’FUNR‘’_REGS-1))) ,\)
translit(
{      }port map(
FOR(BUSNR, 0, NRRBUSES-1,
{ }databus‘’BUSNR => databus‘’BSUBST\BUSNR{,}
{ }srcbus‘’BUSNR => srcbus‘’BSUBST\BUSNR{,}
){ }glock => glock{,}
{ }ck => ck,
{ }data => intern_rdata,
{ }opcode => intern_ropcode);
,\)

   internals:rflfui‘’FUNR
      port map(
tdata => intern_tdata,
topcode => intern_topcode,
tdataact => intern_tdataact,
rdata => intern_rdata,
ropcode => intern_ropcode,
glock => glock,
ck => ck);

end func;
-------------------------------------------------------------------------------
--  Register File FU Internals
--
-- ‘{parameter dependencies: BUSWIDTH PRFLFU._REGS RFLBYPASS}’
-------------------------------------------------------------------------------

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

entity rflfui‘’FUINR is
   port(tdata : in std_logic_vector(BUSWIDTH-1 downto 0);
topcode : in std_logic_vector(_log(_eval(PRFLFU‘’FUINR‘’_REGS-1))-1 downto 0);
tdataact : in std_logic;
rdata : out std_logic_vector(BUSWIDTH-1 downto 0);
ropcode : in std_logic_vector(_log(_eval(PRFLFU‘’FUINR‘’_REGS-1))-1 downto 0);
glock : in std_logic;
ck : in std_logic);
end rflfui‘’FUINR;

architecture func of rflfui‘’FUINR is

   type reg is array (BUSWIDTH-1 downto 0) of std_logic;
   type rfile is array (_eval(PRFLFU‘’FUINR‘’_REGS)-1 downto 0) of reg;

   signal rfl : rfile;
   signal from_rfl : std_logic_vector(BUSWIDTH-1 downto 0);
ifdef({RFLBYPASS},
{   signal rreg : std_logic_vector(BUSWIDTH-1 downto 0);},
{   signal ropcodereg : std_logic_vector(_log(_eval(PRFLFU‘’FUINR‘’_REGS-1))-1 downto 0);})
begin
```

```
   process
   begin
      wait until ck'event and ck='1';
      if (tdataact = '1') then
 rfl(conv_integer(unsigned(topcode))) <= reg(tdata);
      end if;
   end process;

ifdef({RFLBYPASS},
{   from_rfl <= std_logic_vector(rfl(conv_integer(unsigned(ropcode))));
   process
   begin
      wait until ck'event and ck='1';
      if not(glock = '1') then
 if (topcode = ropcode) then
    rreg <= tdata;
 else
    rreg <= from_rfl;
 end if;
      end if;
   end process;
   rdata <= rreg;},
{   process
   begin
      wait until ck'event and ck='1';
      if not(glock = '1') then
 ropcodereg <= ropcode;
      end if;
   end process;
   from_rfl <= std_logic_vector(rfl(conv_integer(unsigned(ropcodereg))));
   rdata <= from_rfl;})
end func;
-------------------------------------------------------------------------------
-- Shift FU
--
-- `{parameter dependencies: PARMOVES BUSWIDTH connection defines}'
-------------------------------------------------------------------------------
library IEEE;
use IEEE.Std_Logic_1164.all;

entity shifu`'FUNR is
   port(
FOR(BUSNR, 0, PARMOVES-1,
--
_ifelse(_eval(CSHIFU`'FUNR`'_TBUS`'BUSNR &&
(CSHIFU`'FUNR`'_R1BUS`'BUSNR || CSHIFU`'FUNR`'_R2BUS`'BUSNR)), 1,
databus`'BUSNR : inout std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval(CSHIFU`'FUNR`'_TBUS`'BUSNR &&
!(CSHIFU`'FUNR`'_R1BUS`'BUSNR || CSHIFU`'FUNR`'_R2BUS`'BUSNR)), 1,
databus`'BUSNR : in std_logic_vector(eval(BUSWIDTH-1) downto 0);,
_eval(!CSHIFU`'FUNR`'_TBUS`'BUSNR &&
(CSHIFU`'FUNR`'_R1BUS`'BUSNR || CSHIFU`'FUNR`'_R2BUS`'BUSNR)), 1,
databus`'BUSNR : out std_logic_vector(eval(BUSWIDTH-1) downto 0);)
_ifelse(_eval(CSHIFU`'FUNR`'_TBUS`'BUSNR ||
CSHIFU`'FUNR`'_R1BUS`'BUSNR || CSHIFU`'FUNR`'_R2BUS`'BUSNR), 1,
squash`'BUSNR : in std_logic;)
_ifelse(CSHIFU`'FUNR`'_TBUS`'BUSNR, 1,
dstbus`'BUSNR : in std_logic_vector(DSTSIZE`'BUSNR-1 downto 0);)
_ifelse(_eval(CSHIFU`'FUNR`'_R1BUS`'BUSNR || CSHIFU`'FUNR`'_R2BUS`'BUSNR), 1,
srcbus`'BUSNR : in std_logic_vector(SRCSIZE`'BUSNR-1 downto 0);)
_ifelse( CSHIFU`'FUNR`'_R1BUS`'BUSNR,
1, r1llockreq`'BUSNR : out std_logic;)
_ifelse( CSHIFU`'FUNR`'_R2BUS`'BUSNR,
1, r2llockreq`'BUSNR : out std_logic;)
)--
texcreq : out std_logic;
r1excreq : out std_logic;
r2excreq : out std_logic;
glock : in std_logic;
outvalid : out std_logic;
valen : in std_logic;

ck : in std_logic;
rst : in std_logic);
end shifu`'FUNR;


use work.sockets.all;
use work.internals.all;

architecture func of shifu`'FUNR is
define({NRTBUSES}, translit(NSHIFU\FUNR\_T,\))
define({NRR1BUSES}, translit(NSHIFU\FUNR\_R1,\))
define({NRR2BUSES}, translit(NSHIFU\FUNR\_R2,\))

   for tsock:insknb`'NRTBUSES use entity WORK.insknb`'NRTBUSES`'(ARCHSTYLE);
   for r1sock:outsnp`'NRR1BUSES use entity WORK.outsnp`'NRR1BUSES`'(ARCHSTYLE);
```

```
   for r2sock:outsnp`'NRR2BUSES use entity WORK.outsnp`'NRR2BUSES`'(ARCHSTYLE);
   for fu_internals:shifui use entity WORK.shifui(ARCHSTYLE);
   for valsock:outval use entity WORK.outval(ARCHSTYLE);

   signal intern_tdata : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_topcode : std_logic_vector(LOG(eval(SHIFUTS-1))-1 downto 0);
   signal intern_tcls : std_logic;
   signal intern_toutsels : std_logic;
   signal intern_tdataact : std_logic;
   signal intern_tsquash : std_logic_vector(NRTBUSES-1 downto 0);

   signal intern_r1data : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_r1act : std_logic;
   signal intern_r1emp : std_logic;
   signal intern_r1socketcls : std_logic;
   signal intern_r1outsels : std_logic;
   signal intern_r1llockreq : std_logic_vector(NRR1BUSES-1 downto 0);
   signal intern_r1squash : std_logic_vector(NRR1BUSES-1 downto 0);

   signal intern_r2data : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal intern_r2act : std_logic;
   signal intern_r2emp : std_logic;
   signal intern_r2socketcls : std_logic;
   signal intern_r2outsels : std_logic;
   signal intern_r2llockreq : std_logic_vector(NRR2BUSES-1 downto 0);
   signal intern_r2squash : std_logic_vector(NRR2BUSES-1 downto 0);

begin
   tsock:insknb`'NRTBUSES
define({BSUBST}, translit(BSHIFU\FUNR\_T,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NRTBUSES-1,
{ }ldstsize`'BUSNR => _sub(DSTSIZE`'BSUBST\BUSNR){,}
{ }lowid`'BUSNR => _sub(ASHIFU\FUNR\_TBUS`'BSUBST\BUSNR){,}
){ }logopcodes => LOG(eval(SHIFUTS-1))) ,\)
translit(
{      }port map(
FOR(BUSNR, 0, NRTBUSES-1,
{ }databus`'BUSNR => databus`'BSUBST\BUSNR{,}
{ }dstbus`'BUSNR => dstbus`'BSUBST\BUSNR{,}
){ }excreq => texcreq,
{ }squash => intern_tsquash,
{ }glock => glock,
{ }ck => ck,
{ }data => intern_tdata,
{ }opcode => intern_topcode,
{ }fucls => intern_tcls,
{ }outsels => intern_toutsels,
{ }dataact => intern_tdataact );,\)

translit(
FOR(BUSNR, 0, NRTBUSES-1, {   }intern_tsquash(BUSNR) <= squash`'BSUBST\BUSNR;
),\)

   r1sock:outsnp`'NRR1BUSES
define({BSUBST}, translit(BSHIFU\FUNR\_R1,\))
translit(
{      }generic map(
FOR(BUSNR, 0, NRR1BUSES-2,
{ }lsrcsize`'BUSNR => _sub(SRCSIZE`'BSUBST\BUSNR){,}
{ }lowid`'BUSNR => _sub(ASHIFU\FUNR\_R1BUS`'BSUBST\BUSNR){,}
){ }lsrcsize`'eval(NRR1BUSES-1) =>_sub(SRCSIZE`'BSUBST\eval(NRR1BUSES-1)){,}
{ }lowid`'eval(NRR1BUSES-1) => _sub(ASHIFU\FUNR\_R1BUS`'BSUBST\eval(NRR1BUSES-1))),\)
translit(
{      }port map(
FOR(BUSNR, 0, NRR1BUSES-1,
{ }databus`'BUSNR => databus`'BSUBST\BUSNR{,}
{ }srcbus`'BUSNR => srcbus`'BSUBST\BUSNR{,}
){ }excreq => r1excreq,
{ }llockreq => intern_r1llockreq,
{ }squash => intern_r1squash,
{ }glock => glock,
{ }ck => ck,
{ }data => intern_r1data,
{ }fuact => intern_r1act,
{ }fuemp => intern_r1emp,
{ }socketcls => intern_r1socketcls,
{ }outsels => intern_r1outsels);,\)

translit(
FOR(BUSNR, 0, NRR1BUSES-1, {   }r1llockreq`'BSUBST\BUSNR <= intern_r1llockreq(BUSNR);
{   }intern_r1squash(BUSNR) <= squash`'BSUBST\BUSNR;
),\)

   r2sock:outsnp`'NRR2BUSES
define({BSUBST}, translit(BSHIFU\FUNR\_R2,\))
```

```
translit(
{      }generic map(
FOR(BUSNR, 0, NRR2BUSES-2,
{ }lsrcsize‘’BUSNR => _sub(SRCSIZE‘’BSUBST\BUSNR){,}
{ }lowid‘’BUSNR => _sub(ASHIFU\FUNR\_R2BUS‘’BSUBST\BUSNR){,}
){ }lsrcsize‘’eval(NRR2BUSES-1) =>_sub(SRCSIZE‘’BSUBST\eval(NRR2BUSES-1)){,}
{ }lowid‘’eval(NRR2BUSES-1) => _sub(ASHIFU\FUNR\_R2BUS‘’BSUBST\eval(NRR2BUSES-1))),\)
translit(
{      }port map(
FOR(BUSNR, 0, NRR2BUSES-1,
{ }databus‘’BUSNR => databus‘’BSUBST\BUSNR{,}
{ }srcbus‘’BUSNR => srcbus‘’BSUBST\BUSNR{,}
){ }excreq => r2excreq,
{ }llockreq => intern_r2llockreq,
{ }squash => intern_r2squash,
{ }glock => glock,
{ }ck => ck,
{ }data => intern_r2data,
{ }fuact => intern_r2act,
{ }fuemp => intern_r2emp,
{ }socketcls => intern_r2socketcls,
{ }outsels => intern_r2outsels);,\)

translit(
FOR(BUSNR, 0, NRR2BUSES-1, {   }r2llockreq‘’BSUBST\BUSNR <= intern_r2llockreq(BUSNR);
{   }intern_r2squash(BUSNR) <= squash‘’BSUBST\BUSNR;
),\)

   fu_internals:shifui
{      }port map(
{ }tdata => intern_tdata,
{ }topcode => intern_topcode,
{ }tcls => intern_tcls,
{ }toutsels => intern_toutsels,
{ }tdataact => intern_tdataact,

{ }r1data => intern_r1data,
{ }r1act => intern_r1act,
{ }r1emp => intern_r1emp,
{ }r1socketcls => intern_r1socketcls,
{ }r1outsels => intern_r1outsels,

{ }r2data => intern_r2data,
{ }r2act => intern_r2act,
{ }r2emp => intern_r2emp,
{ }r2socketcls => intern_r2socketcls,
{ }r2outsels => intern_r2outsels,

{ }ck => ck,
{ }rst => rst);

   valsock:outval
      port map(
outvalid => outvalid,
vin => intern_r1act,
enable => valen);

end func;
-----------------------------------------------------------------------------
-- Shift FU combinational logic
--
-- ‘{parameter dependencies: BUSWIDTH}’
-----------------------------------------------------------------------------
library IEEE;
use IEEE.Std_Logic_1164.all;

entity shifuc is
   port(a : in std_logic_vector(BUSWIDTH-1 downto 0);
opc : in std_logic_vector(LOG(eval(SHIFUTS-1))-1 downto 0);
y1 : out std_logic_vector(BUSWIDTH-1 downto 0);
y2 : out std_logic_vector(BUSWIDTH-1 downto 0);
r1socketcls : in std_logic;
r2socketcls : in std_logic;
socketcls : out std_logic;
r1outsels : in std_logic;
r2outsels : in std_logic;
outsels : out std_logic);
end shifuc;


architecture func of shifuc is

begin
   socketcls <= r1socketcls and r2socketcls;
   outsels <= r1outsels or r2outsels;
   process(a,opc)
   begin
```

```
      case opc is
when "00" => y1(BUSWIDTH-2 downto 0) <= a(BUSWIDTH-1 downto 1);
     y1(BUSWIDTH-1) <= a(BUSWIDTH-1);
     y2(BUSWIDTH-3 downto 0) <= a(BUSWIDTH-1 downto 2);
     y2(BUSWIDTH-2) <= a(BUSWIDTH-1);
     y2(BUSWIDTH-1) <= a(BUSWIDTH-1);
when "01" => y1(BUSWIDTH-2 downto 0) <= a(BUSWIDTH-1 downto 1);
     y1(BUSWIDTH-1) <= '0';
     y2(BUSWIDTH-3 downto 0) <= a(BUSWIDTH-1 downto 2);
     y2(BUSWIDTH-1 downto BUSWIDTH-2) <= "00";
when "10" => y1(BUSWIDTH-1 downto 1) <= a(BUSWIDTH-2 downto 0);
     y1(0) <= '0';
     y2(BUSWIDTH-1 downto 2) <= a(BUSWIDTH-3 downto 0);
     y2(1 downto 0) <= "00";
when others => y1 <= "FOR(CNT,1,BUSWIDTH,`-')";
       y2 <= "FOR(CNT,1,BUSWIDTH,`-')";
      end case;
   end process;
end func;
-----------------------------------------------------------------------------
-- Shift FU internals
--
-- `{parameter dependencies: BUSWIDTH }'
-----------------------------------------------------------------------------
library IEEE;
use IEEE.Std_Logic_1164.all;

entity shifui is
   port(
tdata : in std_logic_vector(eval(BUSWIDTH-1) downto 0);
topcode : in std_logic_vector(LOG(eval(SHIFUTS-1))-1 downto 0);
tcls : out std_logic;
toutsels : out std_logic;
tdataact : in std_logic;

r1data : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
r1act : out std_logic;
r1emp : out std_logic;
r1socketcls : in std_logic;
r1outsels : in std_logic;

r2data : out std_logic_vector(eval(BUSWIDTH-1) downto 0);
r2act : out std_logic;
r2emp : out std_logic;
r2socketcls : in std_logic;
r2outsels : in std_logic;

ck : in std_logic;
rst : in std_logic);
end shifui;

use work.stagecontrol.all;
use work.combinational.all;

architecture func of shifui is

   for stagectl0:stagecnb use entity WORK.stagecnb(ARCHSTYLE);
   for stagectl1:stagecnb use entity WORK.stagecnb(ARCHSTYLE);
   for comblogic:shifuc use entity WORK.shifuc(ARCHSTYLE);

   signal intern_outcls : std_logic;
   signal intern_outactive : std_logic;
   signal intern_outempty : std_logic;
   signal intern_incls : std_logic;
   signal intern_ract : std_logic;
   signal intern_remp : std_logic;

   signal stg0_enable : std_logic;
   signal stg1_enable : std_logic;

   signal intern_socketcls : std_logic;
   signal intern_outsels : std_logic;

   signal wired_to_one : std_logic;
   signal wired_to_zero : std_logic;

   signal treg : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal r1reg : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal r2reg : std_logic_vector(eval(BUSWIDTH-1) downto 0);
   signal opcreg : std_logic_vector(LOG(eval(SHIFUTS-1))-1 downto 0);

begin

   stagectl0:stagecnb
port map(inactive  => tdataact,
              inempty   => wired_to_one,
 incls     => intern_incls,
```

```
 ck        => ck,
 rst       => rst,
 socketcls => intern_socketcls,
 enable    => stg0_enable,

 outactive => intern_outactive,
 outempty  => intern_outempty,
 outcls    => intern_outcls);

   stagectl1:stagecnb
port map(inactive  => intern_outactive,
                 inempty   => intern_outempty,
 incls     => intern_outcls,

 ck        => ck,
 rst       => rst,
 socketcls => intern_socketcls,
 enable    => stg1_enable,

 outactive => intern_ract,
 outempty  => intern_remp,
 outcls    => wired_to_one);

   comblogic:shifuc
port map(a => treg,
 opc => opcreg,
 y1 => r1reg,
 y2 => r2reg,
 r1socketcls => r1socketcls,
 r2socketcls => r2socketcls,
 socketcls => intern_socketcls,
 r1outsels => r1outsels,
 r2outsels => r2outsels,
 outsels => intern_outsels);

   wired_to_one <= '1';
   wired_to_zero <= '0';

   r1act <= intern_ract;
   r2act <= intern_ract;
   r1emp <= intern_remp;
   r2emp <= intern_remp;
   tcls <= intern_incls;
   toutsels <= intern_outsels;

   process
   begin
      wait on ck until ck='1';
      if (stg1_enable='1') then
 r1data <= r1reg;
 r2data <= r2reg;
      end if;
      if (stg0_enable='1') then
 treg <= tdata;
 opcreg <= topcode;
      end if;
   end process;
end func;
------------------------------------------------------------------------------
-- Hybrid pipeline stage controller
--
-- parameter dependencies: none
------------------------------------------------------------------------------
library IEEE;
use IEEE.Std_Logic_1164.all;

entity stagec is
   generic(stagedelay : integer);
   port(inactive : in std_logic;
inempty : in std_logic;
inbsy : out std_logic;
incls : out std_logic;

ck : in std_logic;
rst : in std_logic;
socketcls : in std_logic;
enable : out std_logic;

outactive : out std_logic;
outempty : out std_logic;
outbsy : in std_logic;
outcls : in std_logic);
end stagec;


architecture func of stagec is
```

```
   signal inbsys, inclss, outactives : std_logic;
   signal svalid : std_logic_vector(0 to stagedelay-1);
begin
   enable <= not(not inactive or inbsys or (inclss and socketcls));
   inbsys <= svalid(0) and (not outactives or outbsy);
   inclss <= svalid(0) and outcls;
   outactives <= svalid(stagedelay-1);
   outempty <= not svalid(0) and inempty;

   inbsy <= inbsys;
   incls <= inclss;
   outactive <= outactives;
   process
      variable i : integer;
   begin
      wait until ck'event and ck='1';
      if (stagedelay >1) and not((inbsys or (outcls and socketcls)) = '1') then
 for i in 1 to stagedelay-1 loop
    svalid(i) <= '0';
 end loop;
      else
         for i in 1 to stagedelay-1 loop
    svalid(stagedelay-i) <= svalid(stagedelay-i-1);
         end loop;
      end if;
      svalid(0) <= inactive or inbsys or (inclss and socketcls);
      if not(rst = '1') then
 svalid <= (others => '0');
      end if;
   end process;
end func;

library IEEE;
use IEEE.Std_Logic_1164.all;

entity stagecnb is -- no BSY chain
   port(inactive : in std_logic;
inempty : in std_logic;
incls : out std_logic;

ck : in std_logic;
rst : in std_logic;
socketcls : in std_logic;
enable : out std_logic;

outactive : out std_logic;
outempty : out std_logic;
outcls : in std_logic);
end stagecnb;


architecture func of stagecnb is
   signal inclss, outactives, svalid0 : std_logic;
begin
   enable <= not(not inactive or (inclss and socketcls));
   inclss <= svalid0 and outcls;
   outactives <= svalid0;
   outempty <= not svalid0 and inempty;

   incls <= inclss;
   outactive <= outactives;

   process
   begin
      wait until ck'event and ck='1';
      svalid0 <= inactive or (inclss and socketcls);
      if not(rst='1') then
 svalid0 <= '0';
      end if;
   end process;
end func;
------------------------------------------------------------------------------
-- Valid FU
--
-- `{parameter dependencies: PARMOVES connection defines}'
------------------------------------------------------------------------------

library IEEE;
use IEEE.Std_Logic_1164.all;

entity valfu is
   port(
FOR(BUSNR, 0, PARMOVES-1,
_ifelse(_eval(CVALFU_RBUS`'BUSNR), 1,
{ }srcbus`'BUSNR : in std_logic_vector(SRCSIZE`'BUSNR-1 downto 0);)
) valen : out std_logic_vector(NVALFU_R-1 downto 0);
glock : in std_logic;
```

```
ck : in std_logic);
end valfu;


-- Valid FU decoder

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

entity valdec is
   generic(
lsrcsize0 : integer;
lowid0 : integer
FOR(BUSNR, 1, NVALFU_R-1,;
{ }lsrcsize‘’BUSNR : integer;
{ }lowid‘’BUSNR : integer)
);
   port(
FOR(BUSNR, 0, NVALFU_R-1,
{ }srcbus‘’BUSNR : in std_logic_vector(lsrcsize‘’BUSNR-1 downto 0);
) glock : in std_logic;
ck : in std_logic;
valen : out std_logic_vector(NVALFU_R-1 downto 0));
end valdec;


architecture func of valdec is

begin
   process
 FOR(BUSNR, 0, NVALFU_R-1,
{      }variable id‘’BUSNR : std_logic_vector(lsrcsize‘’BUSNR-1 downto 0);
)   begin
      wait until ck’event and ck = ’1’;
      if not(glock = ’1’) then
FOR(BUSNR, 0, NVALFU_R-1,
{  }id‘’BUSNR := std_logic_vector(conv_unsigned(lowid‘’BUSNR, id‘’BUSNR’length));
{  }if (srcbus‘’BUSNR = id‘’BUSNR) then
{     }valen(BUSNR) <= ’1’;
{  }else
{     }valen(BUSNR) <= ’0’;
{  }end if;
{      }end if;
)   end process;
end func;



architecture func of valfu is
   component valdec
   generic(
lsrcsize0 : integer;
lowid0 : integer
FOR(BUSNR, 1, NVALFU_R-1,;
{ }lsrcsize‘’BUSNR : integer;
{ }lowid‘’BUSNR : integer)
);
   port(
FOR(BUSNR, 0, NVALFU_R-1,
{ }srcbus‘’BUSNR : in std_logic_vector(lsrcsize‘’BUSNR-1 downto 0);
) glock : in std_logic;
ck : in std_logic;
valen : out std_logic_vector(NVALFU_R-1 downto 0));
   end component;

   for rdec : valdec use entity work.valdec(ARCHSTYLE);
begin
   rdec : valdec
      generic map(
lsrcsize0 => _sub(SRCSIZE‘’BVALFU_R0),
lowid0 => _sub(AVALFU_RBUS‘’BVALFU_R0)
FOR(BUSNR, 1, NVALFU_R-1,{ ,}
{ }lsrcsize‘’BUSNR => _sub(SRCSIZE‘’_sub(BVALFU_R‘’BUSNR)){,}
{ }lowid‘’BUSNR => _sub(AVALFU_RBUS‘’_sub(BVALFU_R‘’BUSNR))
) )
      port map(
FOR(BUSNR, 0, NVALFU_R-1,
{ }srcbus‘’BUSNR => srcbus‘’_sub(BVALFU_R‘’BUSNR){,}
) glock => glock,
ck => ck,
valen => valen);
end ARCHSTYLE;
```