ICS

TECHNICAL REPORT

Coprocessor Codesign for Programmable Architectures

Prabhat Mishra, Frederic Rousseau, Nikil Dutt, and Alex Nicolau
{pmishra, dutt, nicolau}@ics.uci.edu
frederic.rousseau@imag.fr
http://www.cecs.uci.edu/~aces

UCI-ICS Technical Report #01-13
Dept. of Information and Computer Science
University of California, Irvine, CA 92697

April, 2001

Information and Computer Science
University of California, Irvine

Coprocessor Codesign for Programmable Architectures

Prabhat Mishra [†] pmishra@cecs.uci.edu
Frederic Rousseau [‡] frederic.rousseau@imag.fr
Nikil Dutt [†] dutt@cecs.uci.edu
Alex Nicolau [†] nicolau@cecs.uci.edu

[†] Architectures and Compilers for Embedded Systems (ACES) Laboratory
Center for Embedded Computer Systems, University of California, Irvine, CA 92697, USA
http://www.cecs.uci.edu/~aces

[‡] System Level Synthesis Group
Laboratoire TIMA, 46 av. Felix Viallet, 38031 Grenoble cedex, FRANCE
http://tima-cmp.imag.fr/Homepages/cosmos/SLS.html

Technical Report #01-13
Dept. of Information and Computer Science
University of California, Irvine, CA 92697, USA

April 2001

Abstract

Embedded systems present a tremendous opportunity to customize the designs by exploiting the application behavior. Shrinking time-to-market, coupled with short product lifetimes create a critical need for rapidly explore and evaluate candidate System-on-Chip(SOC) architectures. Recent work on language driven Design Space Exploration (DSE) uses Architecture Description Language (ADL) to capture the processor-memory architecture and generate automatically a software toolkit for that architecture. We present in this report an ADL-based approach to explicitly capture the coprocessor configuration, and perform exploration of the coprocessor architecture along with processor and memory subsystem. We present a set of experiments using our coprocessor-aware ADL to drive the exploration of the TI C6x processor-memory architecture in the presence of coprocessors, demonstrating a range of cost and performance attributes.

Contents

List of Figures

List of Tables

1 Introduction

Programmable embedded systems are composed of processor, memory subsystem and coprocessors. The coprocessor is used to compute specific functionalities, which processor is not capable of doing or cannot perform efficiently. Usually, we add coprocessor next to the processor in order to compute specific operations not available in the processor or too time consuming if this operation is done by the processor . In this case, the processor sends operands to the coprocessor and waits for results, while performing certain computations in parallel. On the other hand, coprocessor can read its operands from memory using classical I/O ports or via local memory that uses DMA to transfer data from main memory. So when designer adds coprocessor, he has to modify the application program to incorporate these memory transfers. Designer also need to schedule the coprocessor task manually. It means that the designer has to optimize data transfer between processor and coprocessor instructions. However, processor memory accesses are known only after compilation, and it is very difficult to optimize these accesses to incorporate coprocessor. If the coprocessor accesses the main memory directly, the problem remains the same because during compilation the instruction scheduler doesn't know when the memory will be available. In this case, the compilation is independent of the memory access of the coprocessor.

The solution we deal with is to insert the coprocessor inside, as a new functional unit. The re-use of the coprocessor is exactly the same. But inserting it in the processor core presents some interesting aspects. First, the instruction scheduler treats this coprocessor as a functional unit, and the scheduler knows all the timings. So the application code could be optimized, even if memory accesses for (or by) the coprocessor are quite long. In fact, the scheduler can take this into account for the scheduling of all the other instructions which need to access the main memory. Efficient cache management could play a very important role in this scenario. Secondly, the designer does not have to modify the application code (explicit I/O, control signals, interrupts etc.) to test it with the coprocessor. In our EXPRESSION [9] framework, the compiler recognizes the call function that coprocessor will execute. At this point the compiler is not able to extract the code sequence from the application program which the coprocessor can execute. We insert call functions to guide the compiler to schedule certain part of the application program for the coprocessor. The designer can test different application programs quickly. Retargetable simulator and compiler generation helps him to explore different processor and coprocessor configurations in the presence of different application programs.

During HW/SW co-design of embedded systems, designers need to decide which part of the functionality can be implemented in coprocessor. However, to justify the use of coprocessor the designers need to perform design space exploration. Designers have multiple choices viz., using a coprocessor to perform a functionality or implement the function in software, or add a functional unit in the processor itself to perform the task, or modify the existing functional unit(s) to support new operation(s). However, the last two options are feasible only when the modification in the processor core is possible without violating area, timing and power constraints. The first one is the only alternative when processor does not support certain operations e.g, division in TI C6x ([20]). To evaluate the effect of using coprocessor, designers need to perform design space exploration. However, to enable rapid design space exploration there is a need for (i) describing

the embedded system (processor, coprocessor, memory subsystem) in higher level of abstraction, and (ii) generating software toolkit (e.g, compiler, simulator, assembler) automatically from the description.

Recent advances in Systems-on-Chip (SOC) technology enable customization of the processor architecture, memory subsystem, and co-processor architecture to a specific application (or application domain) to meet the diverse requirements, e.g, better performance, low power, smaller area, higher code density etc. However, shrinking time-to-market coupled with short product lifetimes create a critical need to rapidly evaluate candidate SOC architectures, and complete both software and hardware implementations in parallel. Rapid Design Space Exploration (DSE) of SOC architectures is critically dependent on a retargetable software toolkit (e.g., compiler, simulator, assembler etc.). Architecture Description Language (ADL) based approaches ([1], [2], [3], [4], [6], [9], [17], [19], [21]) have recently been proposed to support automatic software toolkit generation for the processor-memory architecture, and provide feedback to the designer on the quality of the architecture. While these approaches extensively address processor and memory features, to our knowledge no previous approach allows explicit capture of a coprocessor, and the attendant task of generating software toolkit that fully exploits this coprocessor architecture.

The contribution of this report is the explicit description of a coprocessor along with processor and memory subsystem in EXPRESSION ADL [9], permitting co-exploration of the processor, coprocessor and the memory architecture. We generate coprocessor-aware software toolkit that allows rapid design space exploration.

This report is organized as follows. Section 2 presents related work addressing ADL-driven DSE approaches. Section 3 outlines our approach and the overall flow of our environment. Section 4 describes the coprocessor description in EXPRESSION. Section 5 describes how we generate coprocessor aware software toolkit. Section 6 presents design space exploration experiments using the software toolkit. Section 7 concludes the report.

2 Related Work

Recent approaches on language-driven design space exploration ([1], [2], [3], [13], [6], [17], [19], [21]), use ADLs to capture the processor architecture, generate automatically a software toolkit for that processor, and provide feedback to the designer on the quality of the architecture.

nML [6] has been used by the retargetable code generation environment CHESS [2] to describe DSP and ASIP processors. In ISDL [3], constraints on parallelism are explicitly specified through illegal operation groupings. This could be tedious for complex architectures like DSPs which permit operation parallelism (e.g. Motorola 56K) and VLIW machines with distributed register files (e.g. TI C6X). MIMOLA [13] descriptions are structure-based, and generally very low-level, and laborious to write. MDes [21] allows only a restricted retargetability of the simulator to the HPL-PD processor family. MDes permits the description of the memory system, but is limited to the traditional cache hierarchy. LISA [5] and RADL [18] capture VLIW DSP processors. The approach of [10] uses LISA and SystemC based framework for fast hardware-software co-simulation. An ADL-based approach for processor-memory co-exploration is presented in [15].

While these approaches explicitly capture the processor and memory features to varying de-

grees, to our knowledge, no previous approach allows explicit capture of a coprocessor along with processor and memory specification, and the attendant tasks of generating a software toolkit that fully exploits this coprocessor architecture. In this report we show how to describe coprocessor in EXPRESSION [9] ADL. A coprocessor-aware software toolkit is generated automatically from the EXPRESSION description to enable design space exploration.

3 Our Approach

Figure 1 shows the flow in our approach. In our IP library based Design Space Exploration (DSE) scenario, the designer starts by selecting a set of components from a processor IP library, memory IP library, and coprocessor IP library. Our EXPRESSION Architectural Description Language (ADL) description (containing a mix of such IP components and custom blocks) is then used to generate the information necessary to target both the compiler and the simulator to the specific architecture.

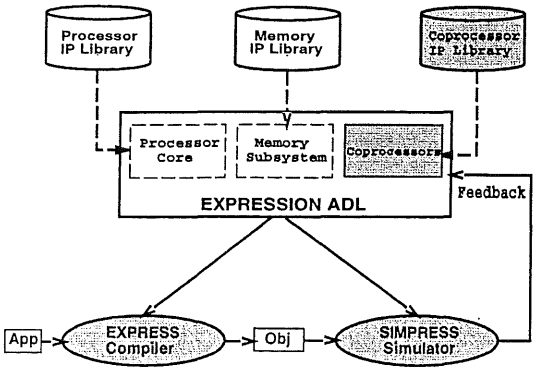

Figure 1. Flow in our approach

Traditionally, a coprocessor is used in embedded system to compute specific operations that processor either cannot perform or do not perform due to performance and other constraints. In this case, the processor send the operation to the coprocessor and wait for the results while computing something else. In certain architectures e.g., ARM [11], the processor does not perform anything while coprocessor is active. In this case, the same operation passes through both processor and coprocessor pipeline; the coprocessor performs the intended task whereas the processor treats the operation as NOP. However, if the operation for the coprocessor is scheduled properly, the functional units in the processor can perform their operations while coprocessor is busy without violating any control or data dependencies. Sometimes the processor reads the operands for the coprocessor operation and send the data to the coprocessor. In this case, the coprocessor behaves as a functional unit in the processor. Usually, coprocessor reads its operand from the memory subsystem. Sometimes it has its own local memory which it uses during computation. The data transfer between coprocessor local memory and main memory is done using DMA controller. While the processor and memory pipeline was captured in detail [15] to allow processor-memory co-exploration, the coprocessor pipeline was not explicitly captured and exploited by the software

toolkit during design space exploration.

We capture coprocessor description in EXPRESSION ADL [9] along with processor and memory description. We generate from this ADL description the EXPRESS compiler and SIMPRESS simulator that can exploit the features available in the coprocessor, allowing for detailed feedback on the coprocessor architecture and its match to the target applications.

4 Coprocessor description in EXPRESSION

In order to explicitly describe the coprocessor in EXPRESSION, we need to capture both the structure and behavior of the coprocessor. We illustrate how we capture coprocessor description in EXPRESSION using the TI C6x [20] architecture. Figure 2 shows a simplified model of the TI C6211 architecture. The pipeline paths are shown using solid lines whereas the data transfer paths are shown using dotted lines. The TI C6211 is an 8-way VLIW DSP processor with a deep pipeline, composed of 4 fetch stages (PG, PS, PR, PW), 2 decode stages (DP, DC), followed by the 8 functional units (L1, S1, M1, D1, L2, S2, M2, D2). In this section we describe how we capture the coprocessor description in EXPRESSION. We first present how to describe coprocessor pipeline along with processor and memory pipelines followed by the description of the instructions supported by the coprocessor.

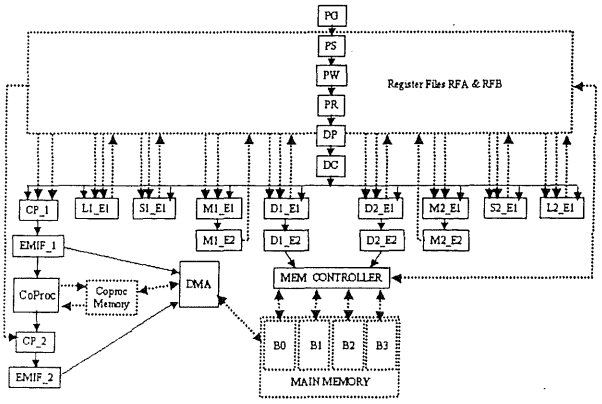

Figure 2. TI C62x Architecture with Coprocessor

4.1 Coprocessor Pipeline

To describe the structure of the coprocessor we capture each pipeline stage of the coprocessor along with their characteristics (e.g, timing, parallelism etc.). The EXPRESSION description for the coprocessor structure (shown in Figure 2) is shown below.

```
;------- Pipeline description --------------------
(PIPELINE PG PS PW PR DP DC Execute)
;Coprocessor appears only in the Execute stage
(Execute (ALTERNATE COPRO L1 S1 M1 D1 D2 M2 S2 L2))
```

```
;Only COPRO pipeline shown here
(COPRO (PIPELINE CP_1 EMIF_1 CoProc CP_2 EMIF_2))
;-------------------------------------------------
;---------------- Characteristics of the CoProc ---
(CoproUnit CoProc
    (CAPACITY 1)
    (TIMING (all 4))
    (OPCODES COPRO_instr)
    (LATCHES .....)
    (PORTS ....)
)
```

The coprocessor pipeline has 5 stages. The coprocessor instruction is decoded in CP_1 stage to determine the size of the input data and the starting address in the main memory. The EMIF_1 stage requests the DMA to transfer the data from the main memory to the coprocessor memory, using efficient access modes if needed. The CoProc stage performs the intended computation (e.g., vector multiply, FFT etc.) using the coprocessor memory for accessing input operands. Results are stored back in the coprocessor memory. The CP_2 stage decides the size of the result and the starting address in the main memory to store the results of the computation, and EMIF_2 requests the DMA to transfer the data from coprocessor memory to main memory. The example above shows the characteristics viz., timing, opcodes supported, parallelism (CAPACITY) for only the CoProc unit. The complete coprocessor description in EXPRESSION has description for remaining four stages (CP_1, EMIF_1, CP_2, EMIF_2), DMA controller, coprocessor memory (in STORAGE section), pipelines latches, ports, connections etc. and can be found in Appendix A.

4.2 Behavior of the Coprocessor

To describe the behavior of the coprocessor we capture the operations it supports. For example, the EXPRESSION description for the vector multiplication operation is shown below.

```
(OP_GROUP COPRO_instr
    (OPCODE VectMul
        (OPERANDS (_SOURCE_1_ mem) (_SOURCE_2_ mem)
                  (_DEST_ mem) (_LENGTH_ immediate)
    )
)
```

Here we show how we capture the vector multiplication operation supported by the coprocessor. Unlike normal instructions whose source and destination operands are register type (except load/store), here source and destination operands are memory type. The _SOURCE_1_ and _SOURCE_2_ fields refer to the starting addresses of two source operands for the multiplication. Similarly _DEST_ refers to the starting address of destination operand for the multiplication. The _LENGTH_ field refers to the vector length of the operation that has immediate data type.

The explicit representation of the coprocessor structure and behavior allows the compiler to exploit the organization of the coprocessor during operation scheduling, and the simulator to provide detailed feedback on the internal coprocessor traffic.

The pipelining and parallelism between the coprocessor operations are described in EXPRESSION through pipeline paths. Pipeline paths represent the ordering between pipeline stages in the architecture (represented as solid lines in Figure 2). For example, a coprocessor operation traverses

first 4 fetch stages (PG, PS, PR, PW) of the processor, followed by the 2 decode stages (DP, DC), and then it goes through 5 coprocessor stages (CP_1, EMIF_1, CoProc, CP_2, EMIF_2). It also traverses the data transfer paths for reading operands and writing results (represented as dotted lines in in Figure 2). For example, a coprocessor read operation traverses CoProc followed by CoProc-Memory which gets the data from main memory using DMA. Thus the pipeline path traverse by the example coprocessor operation is:

```
(PIPELINE PG, PS, PR, PW, DP, DC, CP_1,
          EMIF_1, CoProc, CP_2, EMIF_2)
```

Even though in this example pipeline path is flattened, the pipeline path in EXPRESSION are described in a hierarchical manner. In this manner EXPRESSION can model a variety of coprocessor modules and their characteristics. The EXPRESSION description can be used to drive the generation of both coprocessor-aware compiler and cycle-accurate structural coprocessor simulator, as described in Section 5.

5 Software Toolkit Generation

We generate automatically the software toolkit from the EXPRESSION description of the processor, coprocessor and memory subsystem. In this section we briefly outline how we generate compiler and simulator which exploit the features of the coprocessor. As mentioned earlier we used the EXPRESSION [9] framework for generating retargetable compiler, EXPRESS [8] and simulator, SIMPRESS [12]. In this section we briefly mention how we generate coprocessor-aware simulator and compiler from the ADL description of the architecture.

5.1 Retargetable Simulator Generation

We are able to generate co-processor aware retargetable simulator due to the use of functional abstraction based primitives. We define each stage of the coprocessor unit using parameterized functions. Each function is further composed of generic sub-functions which allows a finer granularity of architectural exploration. For example, the generic CoProc unit uses three sub-functions as shown below. The parameters for the generic functions (e.g., src1Addr, length etc.) are obtained from the EXPRESSION description of the coprocessor as presented in Section 4.

```
CoProc (src1Addr, src2Addr, destAddr, length, funcPtr )
{
    // ReadSrc1, Src1 vector start address and length
    // of the vector are needed to retrieve the values.

    S1 = ReadOperand(src1Addr, length);

    // ReadSrc2
    S2 = ReadOperand(src2Addr, length);

    // Perform the operation
    D0 = ComputeResult(S1, S2, funcPtr);

    // Write the result back.
    WriteOperand(destAddr, length);
}
```

Similarly, we use the generic model of the DMA controller. The detailed description of all the generic abstractions used to retarget the simulator are too long to describe in this report, and can be found in [14].

5.2 Retargetable Compiler Generation

The coprocessor related information available in the EXPRESSION description are used to retarget the compiler. For example, to schedule the operation, the instruction description, and the pipeline and timing information of the coprocessor are used to generate reservation tables [7]. Considering the fact that DMA can take a long time to access (exact timing is known) the main memory, the compiler can schedule operations using reservation tables such that when the coprocessor is busy, the functional units of the processor can execute independent operations. To obtain optimized code of the application the trailblazing percolation scheduling [16] is used.

The on chip memory is shared between coprocessor and the processor core. The data transfer between on chip memory and the coprocessor local memory is performed using DMA controller. The EMIF_1 units sends request to DMA controller to transfer the data from the on chip memory. During DMA transfer the memory controller delays all the load/store requests since we allow only one of them (DMA controller or memory controller) to access the memory. If this delay of the transfer is known before hand, compiler can take this into account during instruction scheduling and generate optimized code. Similarly, DMA controller delays memory access requests when memory controller is accessing the memory. Again, if this delay can be determined statically, compiler can generate better scheduled code.

In this manner we can generate both coprocessor-aware compiler and cycle-accurate structural coprocessor simulator, and thus enable design space exploration and co-design of the coprocessor and processor-memory architecture.

6 Experiments

We present here a set of experiments to show the usefulness of our approach. Our goal is to study the performance impact of using coprocessor to support vector multiplication.

6.1 Experimental Setup

We have used a set of benchmarks from DSPStone fixed point benchmarks that uses vector multiplication. We have chosen TI C62x architecture for the exploration. The M unit of the TI C62x can be used for vector multiplication. The M unit executes the multiplication operation (MUL) iteratively as shown below.

```
// Vector Multiplication code in Application Program
for (i=0; i < n; i++)
    z[i] = a[i] * b[i];

// Translated pseudo assembly that can run on TI C62x
    MOV i, 0
L5: LOAD x, mem_address(a[i])
```

```
    LOAD y, mem_address(b[i])
    MUL  t, x, y
    STORE t, mem_address(z[i])
    INC i
    LT  $cc i n)
    IF $cc L5

// Equivalent instruction for the coprocessor
VectMul(a, b, z, n);
```

However, the same vector multiplication can be performed using a coprocessor. The equivalent coprocessor instruction is shown above. Naturally, the architectural configuration that uses coprocessor is more costly than the other one. In both cases (with or without coprocessor) we described the architecture using EXPRESSION description and generated EXPRESS compiler and SIMPRESS simulator.

6.2 Results

Figure 3 presents a subset of experiments we ran, showing the total cycle counts for the set of benchmarks for two different architectural configuration, viz., with coprocessor and without coprocessor. Table 1 presents the same. The first column presents the benchmarks we ran. The second column presents the number of cycles needed to execute the benchmarks when functional unit is used to perform vector multiplication. The third column presents the number of cycles needed to execute the benchmarks when coprocessor is used to perform vector multiplication. The last column shows the performance improvement obtained due to the coprocessor. The configuration with coprocessor shows 29% improvement for *dotproduct*, vector multiplication dominated DSP kernel whereas it shows only 5% improvement for *fir2dim* since vector multiplication is only a minor part of the program. The performance improvement is due to the fact that coprocessor uses its local memory and rely on efficient DMA transfer. Moreover, functional units (e.g., M1) operate in register-to-register mode whereas co-processor operates on its memory-memory mode. As a result the register pressure and thereby spilling gets reduced in the presence of the coprocessor.

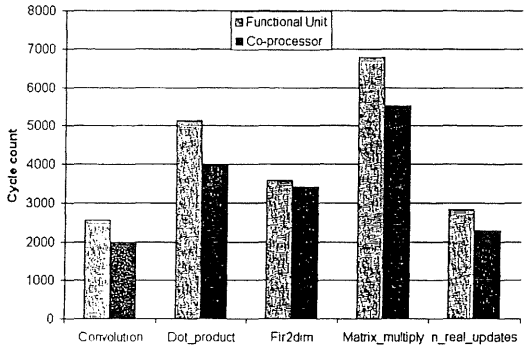

Figure 3. Performance with or without co-processor

We have run another set of experiments for the benchmark *convolution* with vector multiplication

Table 1. Performance Analysis with or without co-processor

Benchmark	Functional Unit	Co-processor	% improvement
convolution	2562	1992	28.61
dot_product	5122	3976	28.82
fir2dim	3581	3407	5.11
matrix_multiply	6786	5526	22.80
n_real_updates	2821	2282	23.62

of different vector length. Table 2 presents the results of this experiment. The first column represents the length of vectors in vector multiplication operation in the *convolution* benchmark. The second column represents the number of cycles needed to execute the program when functional unit is used for vector multiplication. The third column presents the number of cycles needed to execute the program when co-processor is used for performing vector multiplication instead of the functional unit. The last column shows performance improvement for using the coprocessor. Figure 4 presents the same. While the vector length is small (e.g., 1), functional unit performs better since co-processor needs set up cycles. As expected, the coprocessor performs better when vector length is large. The use of coprocessor can deliver upto 29% performance improvement for the *convolution* benchmark.

Table 2. Performance Analysis for *convolution* benchmark for varied vector size

Vector length	Functional Unit	Co-processor	% improvement
1	45	47	-4.26
2	82	74	10.81
4	162	135	20.00
8	322	257	25.29
16	642	504	27.38
32	1282	1000	28.20
64	2562	1992	28.61
128	5122	3976	28.82
256	10242	7944	28.93
512	20482	15880	28.98
1024	40962	31752	29.01

Thus, using our coprocessor-aware ADL-based design space exploration approach, we obtained design points with varying cost and performance. Note that this cannot be determined through analysis alone, the customized coprocessor must be explicitly captured, and the application have to be executed on the configured architecture, as we demonstrated in this section.

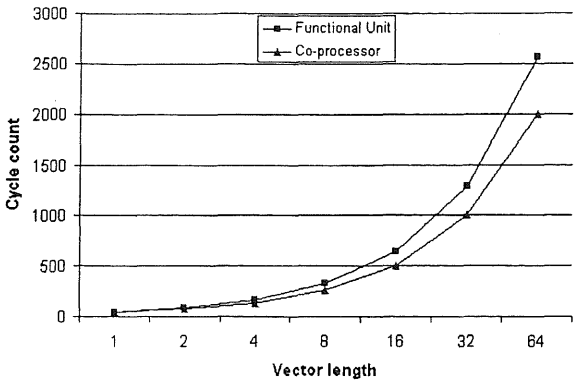

Figure 4. Performance analysis for *convolution* benchmark for varied vector size

7 Summary

This report proposed an ADL driven design space exploration methodology in the presence of coprocessors. We capture the coprocessor description in EXPRESSION along with processor and memory description and generate software toolkit that exploits the coprocessor features. We have demonstrated the power of our approach performing architectural exploration using TI C62x architecture with coprocessor.

Our ongoing work targets the study of cost measures for the local memory, DMA and coprocessor to decide the best cost/performance figure. We plan to perform exploration using larger examples, to study the impact of the application on the coprocessor and overall performance, as well as on system power.

8 Acknowledgments

This work was partially supported by grants from NSF (MIP-9708067), DARPA (F33615-00-C-1632) and a Motorola fellowship. We would like to gratefully acknowledge Ashok Halambi, Peter Grun, and all other EXPRESSION team members for their contribution to the coprocessor codesign work.

References

[1] ARC Cores. *http://www.arccores.com.*

[2] G. G. et al. CHESS: Retargetable code generation for embedded DSP processors. In *Code Generation for Embedded Processors.* Kluwer, 1997.

[3] G. H. et al. ISDL: An instruction set description language for retargetability. In *Proc. DAC,* 1997.

[4] R. L. et al. Retargetable generation of code selectors from HDL processor models. In *Proc. EDTC,* 1997.

[5] V. Z. et al. LISA - machine description language and generic machine model for HW/SW co-design. In *IEEE Workshop on VLSI Signal Processing*, 1996.

[6] M. Freericks. The nML machine description formalism. Technical Report TR SM-IMP/DIST/08, TU Berlin CS Dept., 1993.

[7] P. Grun, A. Halambi, N. Dutt, and A. Nicolau. RTGEN: An algorithm for automatic generation of reservation tables from architectural descriptions. In *ISSS*, San Jose, CA, 1999.

[8] A. Halambi, N. Dutt, and A. Nicolau. Customizing software toolkits for embedded systems-on-chip. In *DIPES 2000*, 2000.

[9] A. Halambi, P. Grun, V. Ganesh, A. Khare, N. Dutt, and A. Nicolau. EXPRESSION: A language for architecture exploration through compiler/simulator retargetability. In *Proc. DATE*, Mar. 1999.

[10] A. Hoffmann, T. Kogel, and H. Meyr. A framework for fast hardware-software co-simulation. In *Proc. DATE*, 2001.

[11] http://www.arm.com/. *ARM7TDMI-S.*

[12] A. Khare, N. Savoiu, A. Halambi, P. Grun, N. Dutt, and A. Nicolau. V-SAT: A visual specification and analysis tool for system-on-chip exploration. In *Proc. EUROMICRO*, 1999.

[13] R. Leupers and P. Marwedel. Retargetable code generation based on structural processor descriptions. *Design Automation for Embedded Systems*, 3(1), 1998.

[14] P. Mishra, J. Astrom, N. Dutt, and A. Nicolau. Functional abstraction of programmable embedded systems. Technical Report UCI-ICS 01-04, University of California, Irvine, 2001.

[15] P. Mishra, P. Grun, N. Dutt, and A. Nicolau. Processor-memory co-exploration driven by an architectural description language. In *Intl. Conf. on VLSI Design 2001*, Bangalore, India, 2001.

[16] A. Nicolau and S. Novack. Trailblazing: A hierarchical approach to percolation scheduling. In *ICPP*, St. Charles, IL, 1993.

[17] V. Rajesh and R. Moona. Processor modeling for hardware software codesign. In *International Conference on VLSI Design*, Jan. 1999.

[18] C. Siska. A processor description language supporting retargetable multi-pipeline dsp program development tools. In *Proc. ISSS*, Dec. 1998.

[19] Tensilica Incorporated. *http://www.tensilica.com.*

[20] Texas Instruments. *TMS320C6201 CPU and Instruction Set Reference Guide*, 1998.

[21] Trimaran Release: http://www.trimaran.org. *The MDES User Manual*, 1997.

A The EXPRESSION ADL description of TI C6x Architecture

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This file contains a specification of the Texas Instruments' TMS326C6201 VLIW + DSP ;
;;;;;;;;; processor, coprocessor, and memory subsystem  in the EXPRESSION ADL. ;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;; Copyright (C) The Regents of the University of California, Irvine ;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;////////////////////Section 1: Specific operations ////////////////

(OPERATIONS_SECTION

  (VAR_GROUPS
    (reg (OR RFA))
    (B14_B15 (OR B14 B15))
    (creg (OR CONTROL_RF))
    (cst21 (OR CONST21))
    (cst16 (OR CONST16))
    (cst5  (OR CONST5))
    (cst5_reg (OR cst5 reg))
    (cst32 (OR CST32))
    (cst15 (OR CONST15))
    (cst21_reg (OR cst21 reg))
    (cst16_reg (OR cst16 reg))
    (addr_mode (OR PREINC PREDEC POSTINC POSTDEC))
    (circ_reg (OR A4_A7 B4_B7)) ;circular addressing modes
    (A4_A7 (OR A4 A5 A6 A7))
    (B4_B7 (OR B4 B5 B6 B7))
    (bk_reg (OR BK0 BK1)) ;block size for circular addressing modes
    (irp (OR IRP))
    (nrp (OR NRP))
  )

  (OP_GROUP single_cycle
    (OPCODE ABS
        (OP_TYPE DATA_OP) (OPERANDS (SRC_1 reg) (DST reg))
    )
    (OPCODE ADD ADDU
        (OP_TYPE DATA_OP) (OPERANDS (SRC_1 reg) (SRC_2 reg) (DST reg))
        (BEHAVIOR "DST=SRC1+SRC2")
    )
    (OPCODE ADDAB ADDAH ADDAW
        (OP_TYPE DATA_OP) (OPERANDS (SRC_1 circ_reg (CIRC bk_reg)) (SRC_2 reg) (DST reg))
        (BEHAVIOR "DST=SRC1+SRC2")
    )
    (OPCODE ADDK
        (OP_TYPE DATA_OP) (OPERANDS (SRC_1 cst16) (DST reg)) (BEHAVIOR "DST=SRC1+DST")
    )
    (OPCODE ADD2 ; SIMD type ADD (upper and lower halves)
        (OP_TYPE DATA_OP) (OPERANDS (SRC_1 reg) (SRC_2 reg) (DST reg))
    )
    (OPCODE AND
        (OP_TYPE DATA_OP) (OPERANDS (SRC_1 reg) (SRC_2 reg) (DST reg))
        (BEHAVIOR "DST=SRC1 & SRC2")
    )
    (OPCODE CLR
        (OP_TYPE DATA_OP) (OPERANDS (SRC_1 reg) (SRC_2 reg) (DST reg))
        (OPERANDS (SRC_1 reg) (SRC_2 cst5) (SRC_3 cst5) (DST reg))
    )
    (OPCODE CMPEQ CMPEQU CMPGT CMPGTU CMPLT CMPLTU
        (OP_TYPE DATA_OP)
        (OPERANDS (SRC_1 cst5_reg) (SRC_2 reg) (DST reg))
    )
```

```
(OPCODE EXT EXTU ; Extract and sign-extend a bit-field
    (OP_TYPE DATA_OP) (OPERANDS (SRC_1 reg) (SRC_2 reg) (DST reg))
    (OPERANDS (SRC_1 reg) (SRC_2 cst5) (SRC_3 cst5) (DST reg))
)
(OPCODE IDLE ; multi-cycle NOP
    (OP_TYPE DATA_OP) (BEHAVIOR none)
)
(OPCODE LMBD ; left most bit detector
    (OP_TYPE DATA_OP) (OPERANDS (SRC_1 cst5_reg) (SRC_2 reg) (DST reg))
)
(OPCODE MV
    (OP_TYPE DATA_OP) (OPERANDS (SRC_1 reg) (DST reg)) (BEHAVIOR "DST=SRC1")
)
(OPCODE MVC ; move from control register file
    (OP_TYPE DATA_OP) (OPERANDS (SRC_1 creg) (DST reg)) (BEHAVIOR "DST=SRC1")
)
(OPCODE MVK
    (OP_TYPE DATA_OP) (OPERANDS (SRC_1 cst16) (DST reg)) (BEHAVIOR "DST=SRC1")
)
(OPCODE MVKH
    (OP_TYPE DATA_OP) (OPERANDS (SRC_1 cst32 MSB16) (DST reg MSB16)) (BEHAVIOR "DST=SRC1")
)
(OPCODE MVKLH
    (OP_TYPE DATA_OP) (OPERANDS (SRC_1 cst32 LSB16) (DST reg MSB16)) (BEHAVIOR "DST=SRC1")
)
(OPCODE NEG
    (OP_TYPE DATA_OP) (OPERANDS (SRC_1 reg) (DST reg)) (BEHAVIOR "DST=-SRC1")
)
(OPCODE NOP
    (OP_TYPE DATA_OP) (OPERANDS none) (BEHAVIOR none)
)
(OPCODE NORM
    (OP_TYPE DATA_OP) (OPERANDS (SRC_1 reg) (DST reg))
)
(OPCODE NOT
    (OP_TYPE DATA_OP) (OPERANDS (SRC_1 reg) (DST reg)) (BEHAVIOR "DST=!SRC1")
)
(OPCODE OR
    (OP_TYPE DATA_OP) (OPERANDS (SRC_1 reg) (SRC_2 reg) (DST reg))
    (BEHAVIOR "DST=SRC1 || SRC2")
)
(OPCODE SADD ; add with saturation
    (OP_TYPE DATA_OP) (OPERANDS (SRC_1 cst5_reg) (SRC_2 reg) (DST reg))
    (BEHAVIOR "DST=SRC1+SRC2")
)
(OPCODE SAT ; saturate a 40bit int into a 32bit one
    (OP_TYPE DATA_OP) (OPERANDS (SRC_1 reg) (DST reg))
)
(OPCODE SET ; set a bit field
    (OP_TYPE DATA_OP) (OPERANDS (SRC_1 reg) (SRC_2 reg) (DST reg))
    (OPERANDS (SRC_1 reg) (SRC_2 cst5) (SRC_3 cst5) (DST reg))
)
(OPCODE SHL
    (OP_TYPE DATA_OP) (OPERANDS (SRC_1 cst5_reg) (SRC_2 reg) (DST reg))
    (BEHAVIOR "DST=SRC2 << SRC1")
)
(OPCODE SHR
    (OP_TYPE DATA_OP) (OPERANDS (SRC_1 cst5_reg) (SRC_2 reg) (DST reg))
    (BEHAVIOR "DST=SRC2 >> SRC1")
)
(OPCODE SHRU
    (OP_TYPE DATA_OP) (OPERANDS (SRC_1 cst5_reg) (SRC_2 reg) (DST reg))
    (BEHAVIOR "DST=SRC2 >> SRC1")
)
```

```
    (OPCODE SSHL ; shift left with saturation
        (OP_TYPE DATA_OP) (OPERANDS (SRC_1 cst5_reg) (SRC_2 reg) (DST reg))
        (BEHAVIOR "DST=SRC2 << SRC1")
    )
    (OPCODE SSUB ; subtraction with saturation
        (OP_TYPE DATA_OP) (OPERANDS (SRC_1 cst5_reg) (SRC_2 reg) (DST reg))
        (BEHAVIOR "DST=SRC1-SRC2")
    )
    (OPCODE SUB SUBU
        (OP_TYPE DATA_OP) (OPERANDS (SRC_1 cst5_reg) (SRC_2 reg) (DST reg))
        (BEHAVIOR "DST=SRC1-SRC2")
    )
    (OPCODE SUBAB SUBAH SUBAW
        (OP_TYPE DATA_OP) (OPERANDS (SRC_1 circ_reg (CIRC bk_reg)) (SRC_2 reg) (DST reg))
        (BEHAVIOR "DST=SRC1-SRC2")
    )
    (OPCODE SUBC ; conditional int subtract with shift (commonly used in division)
        (OP_TYPE DATA_OP) (OPERANDS (SRC_1 cst5_reg LSB16) (SRC_2 reg LSB16) (DST reg))
        (BEHAVIOR "DST=((SRC1 - SRC2) << 1) + 1")
    )
    (OPCODE SUB2 ; SIMD type SUB (upper and lower halves)
        (OP_TYPE DATA_OP) (OPERANDS (SRC_1 reg) (SRC_2 reg) (DST reg))
    )
    (OPCODE XOR
        (OP_TYPE DATA_OP) (OPERANDS (SRC_1 reg) (SRC_2 reg) (DST reg))
        (BEHAVIOR "DST=SRC1 xor SRC2")
    )
    (OPCODE ZERO
        (OP_TYPE DATA_OP) (OPERANDS (DST reg)) (BEHAVIOR "DST=0")
    )
)

(OP_GROUP COPRO_instr
    (OPCODE VectMul
      (OP_TYPE DATA_OP) (OPERANDS (SRC_1 mem) (SRC_2 mem) (DST mem) (LENGTH immediate)
    )
)
(OP_GROUP mpyi_instr
  (OPCODE MPY MPYU MPYUS MPYSU
      (OP_TYPE DATA_OP) (OPERANDS (SRC_1 cst16_reg LSB16) (SRC_2 reg LSB16) (DST reg))
      (BEHAVIOR "DST=SRC1*SRC2")
  )
  (OPCODE MPYH MPYHU MPYHUS MPYHSU
      (OP_TYPE DATA_OP) (OPERANDS (SRC_1 cst5_reg MSB16) (SRC_2 reg MSB16) (DST reg))
      (BEHAVIOR "DST=SRC1*SRC2")
  )
  (OPCODE MPYHL MPYHLU MPYHLUS MPYHLSU
      (OP_TYPE DATA_OP) (OPERANDS (SRC_1 cst5_reg MSB16) (SRC_2 reg LSB16) (DST reg))
      (BEHAVIOR "DST=SRC1*SRC2")
  )
  (OPCODE MPYLH MPYLHU MPYLHUS MPYLHSU
      (OP_TYPE DATA_OP) (OPERANDS (SRC_1 cst5_reg LSB16) (SRC_2 reg MSB16) (DST reg))
      (BEHAVIOR "DST=SRC1*SRC2")
  )
  (OPCODE SMPY ; multiply with shift and saturation
      (OP_TYPE DATA_OP) (OPERANDS (SRC_1 cst5_reg LSB16) (SRC_2 reg LSB16) (DST reg))
      (BEHAVIOR "DST=(SRC1 * SRC2) << 1")
  )
  (OPCODE SMPYHL ; multiply with shift and saturation
      (OP_TYPE DATA_OP) (OPERANDS (SRC_1 cst5_reg MSB16) (SRC_2 reg LSB16) (DST reg))
      (BEHAVIOR "DST=(SRC1 * SRC2) << 1")
  )
  (OPCODE SMPYLH ; multiply with shift and saturation
      (OP_TYPE DATA_OP) (OPERANDS (SRC_1 cst5_reg LSB16) (SRC_2 reg MSB16) (DST reg))
```

```
        (BEHAVIOR "DST=(SRC1 * SRC2) << 1")
    )
    (OPCODE SMPYH ; multiply with shift and saturation
        (OP_TYPE DATA_OP) (OPERANDS (SRC_1 cst5_reg MSB16) (SRC_2 reg MSB16) (DST reg))
        (BEHAVIOR "DST=(SRC1 * SRC2) << 1")
    )
  )
  (OP_GROUP ldst_instr
    (OPCODE LDB LDBU LDH LDHU LDW LDWU
        (OP_TYPE DATA_OP) (OPERANDS (SRC_1 reg addrmode) (SRC_2 cst5_reg) (DST reg))
        (BEHAVIOR "DST=MEM[SRC1+SRC2]")
    )
    (OPCODE LDB15 LDBU15 LDH15 LDHU15 LDW15 LDWU15
        (OP_TYPE DATA_OP) (OPERANDS (SRC_1 B14_B15) (SRC_2 cst15) (DST reg))
        (BEHAVIOR "DST=MEM[SRC1+SRC2]")
    )
    (OPCODE STB STH STW
        (OP_TYPE DATA_OP) (OPERANDS (SRC_1 reg) (SRC_2 reg addrmode) (SRC_3 cst5_reg))
        (BEHAVIOR "MEM[SRC2+SRC3]=SRC1")
    )
    (OPCODE STB15 STH15 STW15
        (OP_TYPE DATA_OP) (OPERANDS (SRC_1 reg) (SRC_2 B14_B15) (SRC_3 cst15))
        (BEHAVIOR "MEM[SRC2+SRC3]=SRC1")
    )
  )
  (OP_GROUP branch_instr
    (OPCODE B
        (OP_TYPE CONTROL_OP) (OPERANDS (SRC_1 cst21_reg))
    )
    (OPCODE BIRP ; branch using interrupt return pointer
        (OP_TYPE CONTROL_OP) (OPERANDS (SRC_1 irp))
    )
    (OPCODE BNRP ; branch using NMI return pointer
        (OP_TYPE CONTROL_OP) (OPERANDS (SRC_1 nrp))
    )
  )
  ; hierarchical groupings of instructions to allow shorter reference from components
  (OP_GROUP L_instr
    (OPCODE ABS ADD ADDU AND CMPEQ CMPGT CMPGTU CMPLT CMPLTU LMBD MV NEG
            NORM NOT OR SADD SAT SSUB SUB SUBU SUBC XOR ZERO
    )
  )
  (OP_GROUP M_instr
    (OPCODE MPY MPYU MPYUS MPYSU MPYH MPHHU MPYHUS MPYHSU MPYHL MPYHLU MPYHULS
            MPYHSLU MPYLH MPYLHU MPYLUHS MPYLSHU SMPY SMPYHL SMPYLH SMPYH
    )
  )
  (OP_GROUP S_instr
    (OPCODE ADD ADDU ADDK ADD2 AND B CLR EXT EXTU MV MVC MVK MVKH MVKLH NEG
            NOT OR SET SHL SHR SHRU SHRL SUB SUBU SUB2 XOR ZERO
    )
  )
  (OP_GROUP S2_instr
    (OPCODE BIRP BNRP
    )
  )
  (OP_GROUP D_instr
    (OPCODE ADD ADDU ADDAB ADDAH ADDAQ LDB LDBU LDH LDHU LDW MV STB STH STW SUB
            SUBAB SUBAH SUBAQ ZERO
    )
  )
  (OP_GROUP D2_instr
    (OPCODE LDB15 LDBU15 LDH15 LDHU15 LDW15 STB15 STH15 STW15
    )
```

```
  )
)

;//////////////////Section 2: Instruction template ////////////////////

(INSTRUCTION_SECTION
    (WORDLEN 32)
    (SLOTS
        ((TYPE DATA) (BITWIDTH 8) (UNIT COPRO))
        ((TYPE DATA) (BITWIDTH 8) (UNIT L1))
        ((TYPE DATA) (BITWIDTH 8) (UNIT L2))
        ((TYPE DATA) (BITWIDTH 8) (UNIT M1))
        ((TYPE DATA) (BITWIDTH 8) (UNIT M2))
        ((TYPE DATA CONTROL) (BITWIDTH 8) (UNIT D1))
        ((TYPE DATA CONTROL) (BITWIDTH 8) (UNIT D2))
        ((TYPE DATA) (BITWIDTH 8) (UNIT S1))
        ((TYPE DATA) (BITWIDTH 8) (UNIT S2))
    )
)

;//////////////Section 3: Operation mappings ////////////////////////////

(OPMAPPING_SECTION
  (OP_MAPPING  (GENERIC (IADD SRC1 SRC2 DST))        (TARGET    (ADD SRC1 SRC2 DST)))
  (OP_MAPPING  (GENERIC (ISUB SRC1 SRC2 DST))        (TARGET (SUB SRC1 SRC2 DST)))
  (OP_MAPPING  (GENERIC (IVLOAD SRC1 SRC2 DST))      (TARGET (LDW SRC1 SRC2 DST)))
  (OP_MAPPING  (GENERIC (IVSTORE SRC1 SRC2 SRC3))    (TARGET (STW SRC1 SRC2 SRC3)))
  (OP_MAPPING  (GENERIC (IASSIGN SRC1 DST))          (TARGET (MV SRC1 DST)))
  (OP_MAPPING  (GENERIC (IASH SRC1 (INT_POZ SRC2) DST))  (TARGET  (SHR SRC1 SRC2 DST)))
  (OP_MAPPING  (GENERIC (IASH SRC1 (INT_NEG SRC2) DST))  (TARGET  (SHL SRC1 (MINUS SRC2) DST)))
  (OP_MAPPING  (GENERIC (ILSH SRC1 (INT_POZ SRC2) DST))  (TARGET  (SHRU SRC1 SRC2 DST)))
  (OP_MAPPING  (GENERIC (ILSH SRC1 (INT_NEG SRC2) DST))  (TARGET  (SHL SRC1 (MINUS SRC2) DST)))
  (OP_MAPPING  (GENERIC (DADD SRC1 SRC2 DST))        (TARGET (ADD SRC1 SRC2 DST)))
  (OP_MAPPING  (GENERIC (DSUB SRC1 SRC2 DST))        (TARGET (SUB SRC1 SRC2 DST)))
  (OP_MAPPING  (GENERIC (DMUL SRC1 SRC2 DST))        (TARGET    (MPY SRC1 SRC2 DST)))
  (OP_MAPPING  (GENERIC (ICONSTANT SRC1 DST))        (TARGET (MVK SRC1 DST)))
  (OP_MAPPING  (GENERIC (ASSIGN SRC1 DST))           (TARGET (MV SRC1 DST)))
  (OP_MAPPING  (GENERIC (IGE SRC1 SRC2 DST))         (TARGET (CMPLT SRC2 SRC1 DST)))
  (OP_MAPPING  (GENERIC (ILT SRC1 SRC2 DST))         (TARGET (CMPLT SRC1 SRC2 DST)))
  (OP_MAPPING  (GENERIC (ILE SRC1 SRC2 DST))         (TARGET (CMPGT SRC2 SRC1 DST)))
  (OP_MAPPING  (GENERIC (IGT SRC1 SRC2 DST))         (TARGET (CMPGT SRC1 SRC2 DST)))
  (OP_MAPPING  (GENERIC (IEQ SRC1 SRC2 DST))         (TARGET (CMPEQ SRC2 SRC1 DST)))
  (OP_MAPPING  (GENERIC (DMTC1 SRC1 DST))            (TARGET (MV SRC1 DST)))
  (OP_MAPPING  (GENERIC (TRUNCID SRC1 DST))          (TARGET (MV SRC1 DST)))
  (OP_MAPPING  (GENERIC (MFC1 SRC1 DST))             (TARGET (MV SRC1 DST)))
  (OP_MAPPING  (GENERIC (DCONSTANT SRC1 DST))        (TARGET (MVK SRC1 DST)))
  (OP_MAPPING  (GENERIC (MTC1 SRC1 DST))             (TARGET (MV SRC1 DST)))
  (OP_MAPPING  (GENERIC (CVTDI SRC1 DST))            (TARGET (MV SRC1 DST)))
  (OP_MAPPING  (GENERIC (ILAND SRC1 SRC2 DST))       (TARGET    (AND SRC1 SRC2 DST)))
  (OP_MAPPING  (GENERIC (DASSIGN SRC1 DST))          (TARGET (MV SRC1 DST)))
  (OP_MAPPING  (GENERIC (DGE SRC1 SRC2 DST))         (TARGET (CMPLT SRC2 SRC1 DST)))
  (OP_MAPPING  (GENERIC (DLT SRC1 SRC2 DST))         (TARGET (CMPLT SRC1 SRC2 DST)))
  (OP_MAPPING  (GENERIC (DLE SRC1 SRC2 DST))         (TARGET (CMPGT SRC2 SRC1 DST)))
  (OP_MAPPING  (GENERIC (DGT SRC1 SRC2 DST))         (TARGET (CMPGT SRC1 SRC2 DST)))
  (OP_MAPPING  (GENERIC (DEQ SRC1 SRC2 DST))         (TARGET (CMPEQ SRC2 SRC1 DST)))
  (OP_MAPPING  (GENERIC (CVTSD SRC1 DST))            (TARGET (MV SRC1 DST)))
  (OP_MAPPING  (GENERIC (CVTDS SRC1 DST))            (TARGET (MV SRC1 DST)))
  (OP_MAPPING  (GENERIC (DMFC1 SRC1 DST))            (TARGET (MV SRC1 DST)))
  (OP_MAPPING  (GENERIC (FADD SRC1 SRC2 DST))        (TARGET    (ADD SRC1 SRC2 DST)))
  (OP_MAPPING  (GENERIC (FSUB SRC1 SRC2 DST))        (TARGET (SUB SRC1 SRC2 DST)))
  (OP_MAPPING  (GENERIC (FMUL SRC1 SRC2 DST))        (TARGET    (MPY SRC1 SRC2 DST)))
  (OP_MAPPING  (GENERIC (FDIV SRC1 SRC2 DST))
    (PREDICATE (POW2 SRC2) (TARGET  (SHR SRC1 (LOG2 SRC2) DST)))
    (PREDICATE TRUE (TARGET    (STW SRC1 SP 0 POSTINC) (STW SRC2 SP 0 POSTINC) (B div)))
```

```
  )
  (OP_MAPPING
    (GENERIC (DDIV SRC1 SRC2 DST))
    (PREDICATE (POW2 SRC2) (TARGET    (SHR SRC1 (LOG2 SRC2) DST)))
    (PREDICATE TRUE (TARGET    (STW SRC1 SP 0 POSTINC) (STW SRC2 SP 0 POSTINC) (B div)))
  )
  (OP_MAPPING    (GENERIC (DVLOAD SRC1 SRC2 DST))    (TARGET (LDW SRC1 SRC2 DST)))
  (OP_MAPPING    (GENERIC (DVSTORE SRC1 SRC2 SRC3))    (TARGET (STW SRC1 SRC2 SRC3)))
  (OP_MAPPING    (GENERIC (FVLOAD SRC1 SRC2 DST))    (TARGET (LDW SRC1 SRC2 DST)))
  (OP_MAPPING    (GENERIC (FVSTORE SRC1 SRC2 SRC3))    (TARGET (STW SRC1 SRC2 SRC3)))
)

;////////////////// Section 4: Components Specification ////////////////////

(ARCHITECTURE_SECTION
 (SUBTYPE UNIT ProgAddrGenUnit ProgAddrSendUnit ProgAccessWaitUnit ProgFetchPktReceiveUnit
     InstrDispatchUnit InstrDecodeUnit CP1phase EMIF1phase CoProcUnit CP2phase EMIF2phase
     LUnitE1phase MUnitE1phase SUnitE1phase DUnitE1phase MUnitE2phase DUnitE2phase MemoryController)
 (SUBTYPE PORT UnitPort Port)
 (SUBTYPE CONNECTION RegisterConnection MemoryConnection)
 (SUBTYPE STORAGE RegFile VirtualMemory Memory)
 (SUBTYPE LATCH OneOpLatch ThreeOpLatch ThreeOpBranchLatch ThreeOpLoadStoreLatch TwoOpLatch ListOpLatch)

 (ProgAddrGenUnit PG
  (CAPACITY 8) (TIMING (all 1)) (OPCODES all) (LATCHES (OUT PGLatch))
 )
 (OneOpLatch PGLatch)

 (ProgAddrSendUnit PS
  (CAPACITY 8) (TIMING (all 1)) (OPCODES all) (LATCHES (OUT PSLatch)) (LATCHES (IN PGLatch))
 )
 (OneOpLatch PSLatch)

 (ProgAccessWaitUnit PW
  (CAPACITY 8) (TIMING (all 1)) (OPCODES all) (LATCHES (OUT PWLatch)) (LATCHES (IN PSLatch))
 )
 (OneOpLatch PWLatch)

 (ProgFetchPktReceiveUnit PR
  (CAPACITY 8) (TIMING (all 1)) (OPCODES all) (LATCHES (OUT PRLatch)) (LATCHES (IN PWLatch))
 )
 (OneOpLatch PRLatch)

 (InstrDispatchUnit DP
  (CAPACITY 8) (TIMING (all 1)) (OPCODES all) (LATCHES (OUT DPLatch)) (LATCHES (IN PRLatch))
 )
 (OneOpLatch DPLatch)

 (InstrDecodeUnit DC
  (CAPACITY 8) (TIMING (all 1)) (OPCODES all)
  (LATCHES (OUT decodeLatch1) (OUT decodeLatch3) (OUT decodeLatch7) (OUT decodeLatch5)
           (OUT decodeLatch6) (OUT decodeLatch8) (OUT decodeLatch4) (OUT decodeLatch2))
  (LATCHES (IN DPLatch))
 )
 (ThreeOpLatch decodeLatch1)
 (ThreeOpLatch decodeLatch3)
 (ThreeOpBranchLatch decodeLatch7)
 (ThreeOpLoadStoreLatch decodeLatch5)
 (ThreeOpLoadStoreLatch decodeLatch6)
 (ThreeOpBranchLatch decodeLatch8)
 (ThreeOpLatch decodeLatch4)
 (ThreeOpLatch decodeLatch2)

 (LUnitE1phase L1_E1
```

```
 (CAPACITY 1) (TIMING (all 1)) (OPCODES all) (LATCHES (IN decodeLatch1))
 (PORTS L1_E1_srcport1 L1_E1_srcport2 L1_E1_dstport3)
)
(UnitPort L1_E1_srcport1("_READ_") (ARGUMENT _SOURCE_1_))
(UnitPort L1_E1_srcport2("_READ_") (ARGUMENT _SOURCE_2_))
(UnitPort L1_E1_dstport3("_WRITE_") (ARGUMENT _DEST_))

(CP1phase CP_1
 (CAPACITY 1) (TIMING (all 1)) (OPCODES COPRO_instr) (LATCHES (IN decodeLatch9))
 (LATCHES (OUT CP1Latch) (PORTS CP_1_srcport1 CP_1_srcport2)
)
(UnitPort CP_1_srcport1("_READ_") (ARGUMENT _SOURCE_1_))
(UnitPort CP_1_srcport2("_READ_") (ARGUMENT _SOURCE_2_))
(ListOpLatch CP1Latch)

(EMIF1phase EMIF_1
 (CAPACITY 1) (TIMING (all 1)) (OPCODES COPRO_instr) (LATCHES (IN CP1Latch))
 (LATCHES (OUT EMIF1Latch) (OUT EMIF1DMALatch))
)
(ListOpLatch EMIF1Latch EMIF1DMALatch)

(CoProcUnit COPRO
 (CAPACITY 1) (TIMING (all 4)) (OPCODES COPRO_instr) (LATCHES (IN EMIF1Latch))
 (LATCHES (OUT CoProcLatch) (PORTS CoProcReadPort1 CoProcWritePort1)
)
(UnitPort CoProcReadPort1("_READ_") (ARGUMENT _SOURCE_1_))
(UnitPort CoProcWritePort1("_WRITE_") (ARGUMENT _DEST_))
(ListOpLatch CoProcLatch)

(CP2phase CP_2
 (CAPACITY 1) (TIMING (all 1)) (OPCODES COPRO_instr) (LATCHES (IN CoProcLatch))
 (LATCHES (OUT CP2Latch) (PORTS CP_2_srcport1)
)
(UnitPort CP_2_srcport1("_READ_") (ARGUMENT _DEST_))
(ListOpLatch CP2Latch)

(EMIF2phase EMIF_2
 (CAPACITY 1) (TIMING (all 1)) (OPCODES COPRO_instr) (LATCHES (IN CP2Latch))
 (LATCHES (OUT EMIF2DMALatch))
)
(ListOpLatch EMIF2DMALatch)

(MUnitE1phase M1_E1
 (CAPACITY 1) (TIMING (all 1)) (OPCODES all) (LATCHES (OUT M1_E1Latch))
 (LATCHES (IN decodeLatch3)) (PORTS M1_E1_srcport1 M1_E1_srcport2)
)
(ThreeOpLatch M1_E1Latch)
(UnitPort M1_E1_srcport1("_READ_") (ARGUMENT _SOURCE_1_))
(UnitPort M1_E1_srcport2("_READ_") (ARGUMENT _SOURCE_2_))

(SUnitE1phase S1_E1
 (CAPACITY 1) (TIMING (all 1)) (OPCODES all) (LATCHES (IN decodeLatch7))
 (PORTS S1_E1_srcport1 S1_E1_srcport2 S1_E1_dstport3)
)
(UnitPort S1_E1_srcport1("_READ_") (ARGUMENT _SOURCE_1_))
(UnitPort S1_E1_srcport2("_READ_") (ARGUMENT _SOURCE_2_))
(UnitPort S1_E1_dstport3("_WRITE_") (ARGUMENT _DEST_))

(DUnitE1phase D1_E1
 (CAPACITY 1) (TIMING (all 1)) (OPCODES all) (LATCHES (OUT D1_E1Latch))
 (LATCHES (IN decodeLatch5)) (PORTS D1_E1_srcport1 D1_E1_srcport2 D1_E1_dstport3)
)
(TwoOpLatch D1_E1Latch)
(UnitPort D1_E1_srcport1("_READ_") (ARGUMENT _SOURCE_1_))
```

```
 (UnitPort D1_E1_srcport2("_READ_") (ARGUMENT _SOURCE_2_))
 (UnitPort D1_E1_dstport3("_WRITE_")
    (OPCODES ADD ADDU ADDAB ADDAQ MV SUB SUBAB SUBAH SUBAW ZERO) (ARGUMENT _DEST_)
 )

 (DUnitE1phase D2_E1
  (CAPACITY 1) (TIMING (all 1)) (OPCODES all) (LATCHES (OUT D2_E1Latch))
  (LATCHES (IN decodeLatch6)) (PORTS D2_E1_srcport2 D2_E1_srcport1 D2_E1_dstport3)
 )
 (TwoOpLatch D2_E1Latch)
 (UnitPort D2_E1_srcport2("_READ_") (ARGUMENT _SOURCE_2_))
 (UnitPort D2_E1_srcport1("_READ_") (ARGUMENT _SOURCE_1_))
 (UnitPort D2_E1_dstport3("_WRITE_")
    (OPCODES ADD ADDU ADDAB ADDAQ MV SUB SUBAB SUBAH SUBAW ZERO) (ARGUMENT _DEST_)
 )

 (SUnitE1phase S2_E1
  (CAPACITY 1) (TIMING (all 1)) (OPCODES all) (LATCHES (IN decodeLatch8))
  (PORTS S2_E1_srcport1 S2_E1_srcport2 S2_E1_dstport3)
 )
 (UnitPort S2_E1_srcport1("_READ_") (ARGUMENT _SOURCE_1_))
 (UnitPort S2_E1_srcport2("_READ_") (ARGUMENT _SOURCE_2_))
 (UnitPort S2_E1_dstport3("_WRITE_") (ARGUMENT _DEST_))

 (MUnitE1phase M2_E1
  (CAPACITY 1) (TIMING (all 1)) (OPCODES all) (LATCHES (OUT M2_E1Latch))
  (LATCHES (IN decodeLatch4)) (PORTS M1_E1_srcport1 M1_E1_srcport2)
 )
 (ThreeOpLatch M2_E1Latch)
 (UnitPort M1_E1_srcport1("_READ_") (ARGUMENT _SOURCE_1_))
 (UnitPort M1_E1_srcport2("_READ_") (ARGUMENT _SOURCE_2_))

 (LUnitE1phase L2_E1
  (CAPACITY 1) (TIMING (all 1)) (OPCODES all) (LATCHES (IN decodeLatch2))
  (PORTS L2_E1_srcport1 L2_E1_srcport2 L2_E1_dstport3)
 )
 (UnitPort L2_E1_srcport1("_READ_") (ARGUMENT _SOURCE_1_))
 (UnitPort L2_E1_srcport2("_READ_") (ARGUMENT _SOURCE_2_))
 (UnitPort L2_E1_dstport3("_WRITE_") (ARGUMENT _DEST_))

 (MUnitE2phase M1_E2
  (CAPACITY 1) (TIMING (all 1)) (OPCODES all) (LATCHES (IN M1_E1Latch)) (PORTS M1_E2_dstport3)
 )
 (UnitPort M1_E2_dstport3("_WRITE_") (ARGUMENT _DEST_))

 (DUnitE2phase D1_E2
  (CAPACITY 1) (TIMING (all 1)) (OPCODES all) (LATCHES (OUT D1_E2Latch)) (LATCHES (IN D1_E1Latch))
 )
 (TwoOpLatch D1_E2Latch)

 (DUnitE2phase D2_E2
  (CAPACITY 1) (TIMING (all 1)) (OPCODES all) (LATCHES (OUT D2_E2Latch)) (LATCHES (IN D2_E1Latch))
 )
 (TwoOpLatch D2_E2Latch)

 (MUnitE2phase M2_E2
  (CAPACITY 1) (TIMING (all 1)) (OPCODES all) (LATCHES (IN M2_E1Latch)) (PORTS M2_E2_dstport3)
 )
 (UnitPort M2_E2_dstport3("_WRITE_") (ARGUMENT _DEST_))


 ;;;;;;;;;;;;;;;;;;;;;;;;; MemController

 ;stage 1, does the memory side communication
```

```
(MemCtrlClass memController_E1
  (PORTS memController_Memport1 memController_Memport2
         memController_Memport3 memController_Memport4) (OPCODES ldst_instr)
)

;stage 2, does the RF side communication
(MemCtrlClass memController_E2
  (PORTS memController_RFAreadport memController_RFBreadport
         memController_RFAwriteport memController_RFBwriteport)
)

;;;;;;;;;;;;;;;;;;;;;;;;;; Main Memory controller (outside the CPU)
(MemCtrlClass mainMemController)
(RegFile RFA
  (PORTS
      RFAreadport1 RFAreadport2 RFAreadport3 RFAreadport4 RFAreadport5 RFAreadport6
      RFAreadport7 RFAreadport8 RFAwriteport1 RFAwriteport2 RFAwriteport3 RFAwriteport4
      RFAwriteport5 RFAreadport9 RFAreadXport)
)
(RegFile RFB
  (PORTS
      RFBreadport1 RFBreadport2 RFBreadport3 RFBreadport4 RFBreadport5 RFBreadport6
      RFBreadport7 RFBreadport8 RFBwriteport1 RFBwriteport2 RFBwriteport3 RFBwriteport4
      RFBwriteport5 RFBreadport9 RFBreadXport)
)
(VirtualMemory OnChipMemory ; this is a "virtual" memory, as seen by
                            ; the CPU, containing 4 banks & controller
  (SUBCOMPONENTS memController_E1 memController_E2 MemoryBlock)
  (PORTS memController_RFAport memController_RFBport)
)
(Memory MemoryBlock ; this is the memory block containing the four banks
  (SUBCOMPONENTS Memory_Bank0 Memory_Bank1 Memory_Bank2 Memory_Bank3)
  (PORTS memport1 memport2 memport3 memport4)
)
(Storage Memory_Bank0 (PORTS memport1))
(Storage Memory_Bank1 (PORTS memport2))
(Storage Memory_Bank2 (PORTS memport3))
(Storage Memory_Bank3 (PORTS memport4))

(Port RFAreadport1 (ARGUMENT _SOURCE_1_))
(Port RFAreadport2 (ARGUMENT _SOURCE_2_))
(Port RFAreadport3 (ARGUMENT _SOURCE_1_))
(Port RFAreadport4 (ARGUMENT _SOURCE_2_))
(Port RFAreadport5 (ARGUMENT _SOURCE_1_))
(Port RFAreadport6 (ARGUMENT _SOURCE_2_))
(Port RFAreadport7 (ARGUMENT _SOURCE_1_))
(Port RFAreadport8 (ARGUMENT _SOURCE_2_))
(Port RFBreadport1 (ARGUMENT _SOURCE_1_))
(Port RFBreadport2 (ARGUMENT _SOURCE_2_))
(Port RFBreadport3 (ARGUMENT _SOURCE_1_))
(Port RFBreadport4 (ARGUMENT _SOURCE_2_))
(Port RFBreadport5 (ARGUMENT _SOURCE_1_))
(Port RFBreadport6 (ARGUMENT _SOURCE_2_))
(Port RFBreadport7 (ARGUMENT _SOURCE_1_))
(Port RFBreadport8 (ARGUMENT _SOURCE_2_))
(Port RFAwriteport1 (ARGUMENT _DEST_))
(Port RFAwriteport2 (ARGUMENT _DEST_))
(Port RFAwriteport3 (ARGUMENT _DEST_))
(Port RFAwriteport4 (ARGUMENT _DEST_))
(Port RFAwriteport5 (ARGUMENT _MEM_SRC_))
(Port RFAreadport9 (ARGUMENT _MEM_DEST_))
(Port RFBwriteport1 (ARGUMENT _DEST_))
(Port RFBwriteport2 (ARGUMENT _DEST_))
(Port RFBwriteport3 (ARGUMENT _DEST_))
```

```
(Port RFBwriteport4 (ARGUMENT _DEST_))
(Port RFBwriteport5 (ARGUMENT _MEM_SRC_))
(Port RFBreadport9 (ARGUMENT _MEM_DEST_))

;RFA, RFB cross connection ports
;the register file ports have to have assigned arguments,
;to be able to map them to the SRC/DST field in the instruction
;these guys go to both source 1 and source 2 of the functional units
(Port RFAreadXport (ARGUMENT _SOURCE_1_)  (ARGUMENT _SOURCE_2_))
(Port RFBreadXport (ARGUMENT _SOURCE_1_)  (ARGUMENT _SOURCE_2_))

;memory controller ports (towards RFs and towards mem)
;ports towards the register files
(Port memController_RFAreadport
  (ARGUMENT _MEM_DEST_) (OPCODES STB STH STW) (TIMING (all 0))
)
(Port memController_RFAwriteport
  (ARGUMENT _MEM_SRC_) (OPCODES LDB LDH LDW) (TIMING (all 0))
)
(Port memController_RFBreadport
  (ARGUMENT _MEM_DEST_) (OPCODES STB STH STW) (TIMING (all 0))
)
(Port memController_RFBwriteport
  (ARGUMENT _MEM_SRC_) (OPCODES LDB LDH LDW) (TIMING (all 0))
)
;ports towards the memory banks
(Port memController_Memport1
  (ARGUMENT _MEM_BYTE_) (TIMING (all 0))
)
(Port memController_Memport2
  (ARGUMENT _MEM_BYTE_) (TIMING (all 0))
)
(Port memController_Memport3
  (ARGUMENT _MEM_BYTE_) (TIMING (all 0))
)
(Port memController_Memport4
  (ARGUMENT _MEM_BYTE_) (TIMING (all 0))
)

;memory ports (4 banks, each with one 16bit port)
(Port memport1("_READWRITE_") (ARGUMENT _MEM_BYTE_))
(Port memport2("_READWRITE_") (ARGUMENT _MEM_BYTE_))
(Port memport3("_READWRITE_") (ARGUMENT _MEM_BYTE_))
(Port memport4("_READWRITE_") (ARGUMENT _MEM_BYTE_))

;L1, L2 connections
(Connection L1_E1_sport1RFA)
(Connection L1_E1_sport2RFA)
(Connection L1_E1_dport3RFA)
(Connection L2_E1_sport1RFB)
(Connection L2_E1_sport2RFB)
(Connection L2_E1_dport3RFB)

;M1, M2 connections
(Connection M1_E1_sport1RFA)
(Connection M1_E1_sport2RFA)
(Connection M1_E2_dport3RFA)
(Connection M2_E1_sport1RFB)
(Connection M2_E1_sport2RFB)
(Connection M2_E2_dport3RFB)

;D1, D2 connections
(Connection D1_E1_sport1RFA)
(Connection D1_E1_sport2RFA)
```

```
    (Connection D1_E1_dport3RFA)
    (Connection D2_E1_sport1RFB)
    (Connection D2_E1_sport2RFB)
    (Connection D2_E1_dport3RFB)

    ;S1, S2 connections
    (Connection S1_E1_sport1RFA)
    (Connection S1_E1_sport2RFA)
    (Connection S1_E1_dport3RFA)
    (Connection S2_E1_sport1RFB)
    (Connection S2_E1_sport2RFB)
    (Connection S2_E1_dport3RFB)

    ; cross connections between opposite register files and units
    (Connection L1_E1_sport1RFB)
    (Connection L1_E1_sport2RFB)
    (Connection M1_E1_sport2RFB)
    (Connection S1_E1_sport2RFB)
    (Connection L2_E1_sport1RFA)
    (Connection L2_E1_sport2RFA)
    (Connection M2_E1_sport2RFA)
    (Connection S2_E1_sport2RFA)

    ; Coprocessor, DMA, coprocessor memory connections
    (Connection CP_1_sport1RFA)
    (Connection CP_1_sport2RFA)
    (Connection CP_2_sport1RFA)
    (Connection CoProc_CoProcMemory)
    (Connection CoProcMemory_CoProc)
    (Connection CoProcMemory_DMA)
    (Connection DMA_OnChipMemory)

    ; memory controller connections
    (Connection memController_Mem1)
    (Connection memController_Mem2)
    (Connection memController_Mem3)
    (Connection memController_Mem4)
    (Connection memController_RFAread)
    (Connection memController_RFAwrite)
    (Connection memController_RFBread)
    (Connection memController_RFBwrite)

    ;; dummy resources to avoid false path in the memory
    (Connection dummy1)
    (Connection dummy2)
    (Connection dummy3)
    (Connection dummy4)
)

;/////////// Section 5: Pipeline and Data-transfer paths ////////////////

(PIPELINE_SECTION
  (PIPELINE PG PS PW PR DP DC Execute)
  (Execute (ALTERNATE COPRO L1_E1 M1 S1_E1 D1 L2_E1 M2 S2_E1 D2))
  (COPRO (PIPELINE CP_1 EMIF_1 CoProc CP_2 EMIF_2))
  (M1 (PIPELINE M1_E1 M1_E2))
  (M2 (PIPELINE M2_E1 M2_E2))
  (D1 (ALTERNATE D1_E1 D1_load D1_store))
  (D2 (ALTERNATE D2_E1 D2_load D2_store))
  (D1_store (PIPELINE D1_E1_MC2 D1_E2_MC1 mainMemController))
  (D1_load (PIPELINE D1_E1 D1_E2 mainMemController memController_E1 memController_E2))
  (D2_store (PIPELINE D2_E1_MC2 D2_E2_MC1 mainMemController))
  (D2_load (PIPELINE D2_E1 D2_E2 mainMemController memController_E1 memController_E2))
  (D1_E1_MC2 (PARALLEL D1_E1 memController_E2))
```

```
(D1_E2_MC1 (PARALLEL D1_E2 memController_E1))
(D2_E1_MC2 (PARALLEL D2_E1 memController_E2))
(D2_E2_MC1 (PARALLEL D2_E2 memController_E1))


;;;;;;;;;;;;;;;;;;; data transfer paths

(DTPATHS
  (TYPE UNI
;;;;;        From To       Path taken ;;;;;;;;;;;
;L1, L2 transfers
         (RFA L1_E1  RFAreadport1 L1_E1_sport1RFA L1_E1_srcport1)
         (RFA L1_E1  RFAreadport2 L1_E1_sport2RFA L1_E1_srcport2)
         (L1_E1 RFA  L1_E1_dstport3 L1_E1_dport3RFA RFAwriteport1)
         (RFB L2_E1  RFBreadport1 L2_E1_sport1RFB L2_E1_srcport1)
         (RFB L2_E1  RFBreadport2 L2_E1_sport2RFB L2_E1_srcport2)
         (L2_E1 RFB  L2_E1_dstport3 L2_E1_dport3RFB RFBwriteport1)

;M1, M2 transfers
         (RFA M1_E1  RFAreadport3 M1_E1_sport1RFA M1_E1_srcport1)
         (RFA M1_E1  RFAreadport4 M1_E1_sport2RFA M1_E1_srcport2)
         (M1_E2 RFA  M1_E2_dstport3 M1_E2_dport3RFA RFAwriteport2)
         (RFB M2_E1  RFBreadport3 M2_E1_sport1RFB M2_E1_srcport1)
         (RFB M2_E1  RFBreadport4 M2_E1_sport2RFB M2_E1_srcport2)
         (M2_E2 RFB  M2_E2_dstport3 M2_E2_dport3RFB RFBwriteport2)

;D1, D2 transfers
         (RFA D1_E1  RFAreadport5 D1_E1_sport1RFA D1_E1_srcport1)
         (RFA D1_E1  RFAreadport6 D1_E1_sport2RFA D1_E1_srcport2)
         (D1_E1 RFA  D1_E1_dstport3 D1_E1_dport3RFA RFAwriteport3)
         (RFB D2_E1  RFBreadport5 D2_E1_sport1RFB D2_E1_srcport1)
         (RFB D2_E1  RFBreadport6 D2_E1_sport2RFB D2_E1_srcport2)
         (D2_E1 RFB  D2_E1_dstport3 D2_E1_dport3RFB RFBwriteport3)

;S1, S2 transfers
         (RFA S1_E1  RFAreadport7 S1_E1_sport1RFA S1_E1_srcport1)
         (RFA S1_E1  RFAreadport8 S1_E1_sport2RFA S1_E1_srcport2)
         (S1_E1 RFA  S1_E1_dstport3 S1_E1_dport3RFA RFAwriteport4)
         (RFB S2_E1  RFBreadport7 S2_E1_sport1RFB S2_E1_srcport1)
         (RFB S2_E1  RFBreadport8 S2_E1_sport2RFB S2_E1_srcport2)
         (S2_E1 RFB  S2_E1_dstport3 S2_E1_dport3RFB RFBwriteport4)

; cross transfers between opposite register files and units
         (RFB L1_E1  RFBreadXport L1_E1_sport1RFB L1_E1_srcport1)
         (RFB L1_E1  RFBreadXport L1_E1_sport2RFB L1_E1_srcport2)
         (RFB M1_E1  RFBreadXport M1_E1_sport2RFB M1_E1_srcport2)
         (RFB S1_E1  RFBreadXport S1_E1_sport2RFB S1_E1_srcport2)
         (RFA L2_E1  RFAreadXport L2_E1_sport1RFA L2_E1_srcport1)
         (RFA L2_E1  RFAreadXport L2_E1_sport2RFA L2_E1_srcport2)
         (RFA M2_E1  RFAreadXport M2_E1_sport2RFA M2_E1_srcport2)
         (RFA S2_E1  RFAreadXport S2_E1_sport2RFA S2_E1_srcport2)

;CP_1 transfers
         (RFA CP_1  RFAreadport10 CP_1_sport1RFA CP_1_srcport1)
         (RFA CP_1  RFAreadport11 CP_1_sport2RFA CP_1_srcport2)

;CP_2 transfer
         (RFA CP_2  RFAreadport12 CP_2_sport1RFA CP_2_srcport1)
  )

  (TYPE BI
         ; memory controller transfers towards the memory
    (memController_E1 Memory_Bank0  memController_Memport1 dummy1 memController_Mem1 memport1)
    (memController_E1 Memory_Bank1  memController_Memport2 dummy2 memController_Mem2 memport2)
```

```
      (memController_E1 Memory_Bank2  memController_Memport3 dummy1 memController_Mem3 memport3)
      (memController_E1 Memory_Bank3  memController_Memport4 dummy2 memController_Mem4 memport4)
      ; CoProcMemory and DMA transfers
      (CoProcMemory DMA CoProcMemoryPort1 CoProcMemory_DMA DMAPort1)
      (DMA OnChipMemory DMAPort2 DMA_OnChipMemory OnChipMemoryPort1)
    )
    (TYPE UNI
    ; memory controller transfers towards the RFs
      (memController_E2 RFA  memController_RFAwriteport dummy3 memController_RFAwrite RFAwriteport5)
      (RFA memController_E2  RFAreadport9 memController_RFAread dummy4 memController_RFAreadport)
      (memController_E2 RFA  memController_RFBwriteport dummy3 memController_RFBwrite RFBwriteport5)
      (RFA memController_E2  RFBreadport9 memController_RFBread dummy4 memController_RFBreadport)
      ; CoProc and coprocessor memory transfers
      (CoProc CoProcMemory CoProcReadPort1 CoProc_CoProcMemory CoProcMemoryReadPort1)
      (CoProcMemory CoProc CoProcMemoryWritePort1 CoProcMemory_CoProc CoProcWritePort1)
    )
  )
)

  ;/////////////// Section 6: Memory hierarchy ////////////////////////

  (STORAGE_SECTION
    (RFA
        (TYPE REGFILE) (SIZE 32) (WIDTH 32)
    )
    (RFB
        (TYPE REGFILE) (SIZE 16) (WIDTH 32)
    )
    (CoProcMemory
        (ACCESS_PORTS
            (CoProcMemoryReadPort1 (ACCESS_WIDTHS 8 16 32 64 128 256 512 1024))
            (CoProcMemoryWritePort1 (ACCESS_WIDTHS 8 16 32 64 128 256 512 1024))
        )
        (SIZE 2048) (WIDTH 8) (ADDRESS_RANGE (0 65535)) (ACCESS_TIMES 1)
    )
    (OnChipMemory ; contains all the 4 banks
        (ACCESS_PORTS
            (memController_RFAreadport (ACCESS_WIDTHS 8 16 32))
            (memController_RFBreadport (ACCESS_WIDTHS 8 16 32))
            (memController_RFAwriteport (ACCESS_WIDTHS 8 16 32))
            (memController_RFBwriteport (ACCESS_WIDTHS 8 16 32))
            (OnChipMemoryPort1 (ACCESS_WIDTHS 8 16 32 64 128 256 1024))
        )
        (SIZE 65536) (WIDTH 8) (ADDRESS_RANGE (0 65535)) (ACCESS_TIMES 3)
    )
    (Memory_Bank0
        (TYPE SRAM) (SIZE 16384)  (WIDTH 8)  (ACCESS_TIMES 2)
        (ADDRESS_RANGE (0 16376 (STRIDE 8)) (1 16377 (STRIDE 8)))
    )
    (Memory_Bank1
        (TYPE SRAM) (SIZE 16384) (WIDTH 8) (ACCESS_TIMES 2)
        (ADDRESS_RANGE (2 16378 (STRIDE 8)) (3 16379 (STRIDE 8)))
    )
    (Memory_Bank2
        (TYPE SRAM) (SIZE 16384) (WIDTH 8) (ACCESS_TIMES 2)
        (ADDRESS_RANGE (4 16380 (STRIDE 8)) (5 16381 (STRIDE 8)))
    )
    (Memory_Bank3
        (TYPE SRAM) (SIZE 16384) (WIDTH 8) (ACCESS_TIMES 2)
        (ADDRESS_RANGE (6 16382 (STRIDE 8)) (7 16383 (STRIDE 8)))
    )
  )
)
```