# Single-ISA Heterogeneous MAny-core Computer (SHMAC) Project Plan

EECS

14. March 2014

**Abstract**

Energy efficiency is becoming the main design constraint from the largest to the smallest computing systems. For high-performance systems, this is exemplified by the Dark Silicon effect. This is the observation that only a portion of the transistors on a chip can be powered at same time for a given power budget and a constant die area. The reason is that it is becoming increasingly harder to leverage the benefits of production technology improvements without increasing the power consumption. We expect that this situation will be handled by creating heterogeneous computer architectures that contain a collection of processing elements with different power and performance characteristics. Then, only the subset of processing elements that maximize energy efficiency for the current application is enabled. Similar challenges arise from the small battery powered embedded systems where energy efficiency is of paramount importance. Heterogeneous platforms with processing elements with different energy/performance characteristics are needed for optimized implementation of the compute intensive applications envisioned in modern portable equipment, e.g., in biomedical and mobile domains. To investigate the implications and opportunities of this new design paradigm we propose the Single-ISA Heterogeneous MAny-core Computer (SHMAC). SHMAC is an infrastructure for realizing heterogeneous computing systems from a collection of diverse processing elements based on a common high-level architecture. In addition to developing the SHMAC system, we plan to investigate possible solutions to the main system software challenges as well as how near- or sub-threshold technology and system scenario based optimization can be applied to heterogeneous computing systems.

# Contents

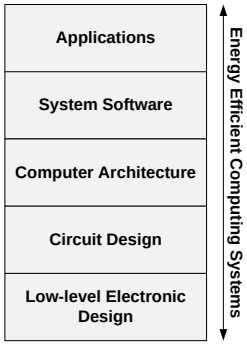


Figure 1: Computing System Abstraction Layers

# 1 Introduction

Computing system performance has grown at an impressive rate since the mid 80s. In the late 80s and 90s, this growth was provided by improvements in production technology, architectural techniques that exploit *Instruction Level Parallelism (ILP)* (e.g. deep pipelines and out-of-order execution) and memory system techniques (e.g. caching) [15]. For the last decade, the performance improvements have been achieved through adding more cores to the chip. The shift to multi-cores were motivated by two observations [19]. First, the amount of ILP that can be extracted from the sequential instruction stream by practical hardware is limited. Second, the hardware complexity grows quickly with the number of instructions that can be executed in parallel. This overhead reduces the energy efficiency of processors that aim to aggressively exploit ILP. Consequently, industry and academia turned to Thread Level Parallelism (TLP) for continued performance growth.

The impressive performance improvement in the 80s and 90s was in part made possible because production technology adhered to the principle called Dennard scaling [9]. In a Dennard scaling process, transistor sizes are reduced while keeping the electric fields constant. This results in a lower power consumption per transistor which again was used to add more transistors to the same die area. The result is more transistors on a fixed chip area at a constant power consumption.

Dennard scaling keeps the electric fields constant by reducing the transistors' threshold voltage. Unfortunately, sub-threshold leakage currents increase exponentially when the threshold voltage is reduced [22] which has lead to the end of Dennard scaling [12]. At the same time, Moore's law is expected to continue for a few generations giving architects an exponentially increasing number of transistors. Moore's law without Dennard scaling creates the Dark Silicon effect [12].

The Dark Silicon effect can be mitigated by two main strategies [6]. First, multi-core architectures can be made *heterogeneous* by including processing elements capable of general computation that have different performance and power characteristics. The task at hand is to select the processing elements that will maximize performance for the current application under a fixed power budget (i.e. maximizing energy efficiency). The remaining processing elements are powered off. Second, architects can add processing elements that are not capable of general computation but very energy efficient for their task. If this task does not occur, the unit is turned off. This technique is often referred to as *specialization* and the specialized unit is called an *accelerator*.

The Dark Silicon effect leads to a paradigm shift in the design of computing systems which effects all abstraction layers (shown in Figure 1) and results in many open research questions. For general systems, system software will need to handle much more diverse processing resources that has been the case in the past [3]. For systems where some information of the application is available, the system can be optimized both at design time and runtime with a system scenario approach [13]. At the architectural level, it is still unclear which processing elements and accelerators that should be included. At the circuit and transistor level, sub- and near-threshold technologies offer to significantly reduce power consumption which means that more transistors can be working simultaneously under the power budget.

To investigate these and other issues of heterogeneous processing systems, we propose the Single-ISA Heterogeneous MAny-core Computer (SHMAC) [20]. SHMAC is an infrastructure for investigating heterogeneous systems at all abstraction levels. The key idea is to create a flexible framework in which different heterogeneous processors can be created from a collection of processing elements and accelerators. To facilitate software design space exploration, the programming model is kept constant across SHMAC-instances but the underlying implementation changes. SHMAC is a tile-based architecture with a mesh interconnect [5, 24]. To achieve a common programming model, all processor tiles implement the ARM ISA and the same memory model. The SHMAC architecture and programming model is discussed in detail in Section 2.

We believe that the SHMAC-approach gives us the right tools to reach the research goals outlined in this plan. For software research, SHMAC makes it possible to explore substantially more diverse systems than those currently provided by the industry. At the same time, SHMAC-architectures realized in FPGAs will be significantly faster than simulator-based approaches. We expect that co-developing software and hardware result in substantial cross-fertilization that gives insights into both hardware and software issues. Finally, the SHMAC micro- and macro-architecture components can be combined with novel transistor technologies and ASIC realizations to reach research goals at the lower abstraction levels.

The SHMAC project is divided into 4 work packages (WPs) which are discussed in detail in Section 3. WP1 is the main development track of SHMAC while WP2, WP3 and WP4 investigates system software, sub- and near-threshold technology and system scenarios, respectively. WP1 delivers new versions of the SHMAC system to the other work packages in 12 month development cycles.

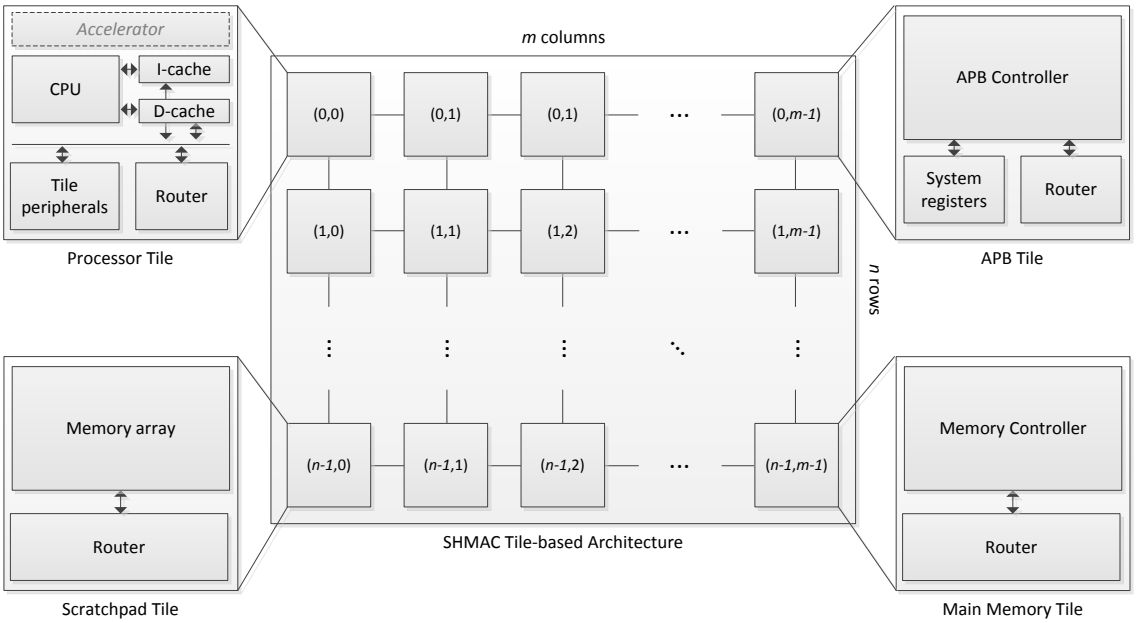


Figure 2: High-Level Architecture of ARM-based SHMAC

# 2 The SHMAC Architecture

SHMAC is a tile-based architecture [5, 24]. In a tile-based architecture, processing elements are laid out in a rectangular grid with connections to their nearest neighbor. We identify each tile by its position in the grid on the form $(n,m)$ where $n$ gives the row and $m$ gives the column. SHMAC uses a mesh interconnect which means that a middle tile (e.g. (1,1)) will have connections to its north, east, south and west neighbors. A border tile (e.g. (0,0)) will only have connections in the directions where there are neighbors. Figure 2 shows the high-level architecture of SHMAC.

SHMAC currently supports the following tile types:

- *Processor Tile*: Processor tiles contain a processor, caches, peripherals and optional accelerators.
- *Scratchpad Tile*: This tile includes a memory and a router but no processing element. A scratchpad [2] is an on-chip memory where the contents are managed by software (e.g. programmer, compiler, system software, etc.).
- *Main Memory Tile*: The Main Memory Tile contains a memory controller that gives SHMAC access to off-chip memory.
- *APB Interface Tile*: This tile implements the Advanced Peripherals Bus (APB) slave which gives the host processor on the Versatile boards access to SHMACs memory space. In addition, the APB contains *system registers* which are used for managing communication with the host system.
- *Dummy Tile*: This tile only contains a router and is used to fill remaining tiles when there is not enough resources available in the target FPGA to fill all tiles with tiles that implement functionality.

The SHMAC single ISA ambition requires that all processing elements can functionally execute the whole ISA. However, the performance will differ based on the available resources. We plan to support this behavior by utilizing the processors exception mechanism which is the standard technique for handling unimplemented instructions [15]. The attempt to execute an unimplemented instruction results in a jump to an exception handling routine that uses the available instructions to execute the functionality of the unimplemented instruction.

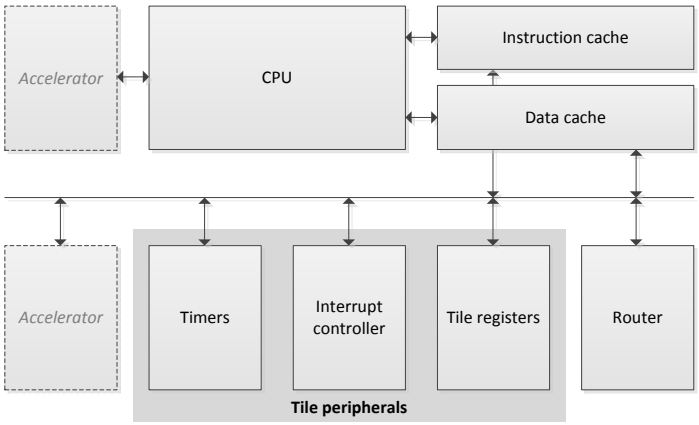


Figure 3: SHMAC Processor Tile

## 2.1 SHMAC Processor Tiles

All processor tiles are able to execute the complete ARM instruction set. Figure 3 shows a block diagram of a processor tile. Each processor tile has a processor core, instruction and data caches, a router and *tile peripherals*. Currently, the tile peripherals contain an interrupt controller, timers and *tile registers*. The tile registers are per-tile memory that store local information like the processor ID, the coordinates of the tile, etc. The caches, router and peripherals are connected to a Wishbone bus.

Caches are added to reduce the average memory latency. However, adding caches also add the possibility for coherence problems. Currently, coherency issues are handled in software by making shared data reside in uncacheable memory regions.

The processor tiles can be equipped with optional accelerators. These accelerators are chosen specifically to be very efficient for a given computation. Since the processor implements the complete instruction set, the accelerators can be optimized for their intended computation while the tile as a whole still is able to functionally execute the complete ISA.

## 2.2 SHMAC Memory Map

Figure 4 shows SHMAC's memory map. The Exception Table is a collection of jump instructions that are used to handle various exceptions. It starts at address 0 and contains 8 instructions, one for each exception type. The high end of the memory space is reserved for system registers and tile registers. The tile registers are private for each tile and contain information like the tiles coordinates and other useful data while the system registers are used for communication with the host system.

The scratchpad tiles have been given a 128 MB address space. The SHMAC architecture supports 1 to $k$ scratchpad tiles where $k$ can be chosen independently of the number of tiles in the SHMAC instance. The number of scratchpad tiles depends on a number of factors like the amount of block RAM available on the chosen FPGA, the desired access latency, the amount of block RAM used for caches in the processor tiles and the possibility of access contention.

## 2.3 SHMAC Interconnect Architecture

Since SHMAC implements a tiled architecture, a 2D mesh-based Network-on-Chip (NoC) interconnect was chosen to provide efficient on-chip and off-chip communication. The parameters of the interconnect are summarized in Table 1.

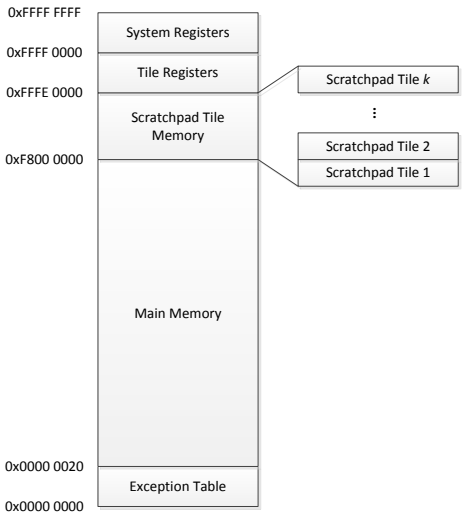


Figure 4: Memory Map of ARM-based SHMAC

Table 1: Architectural parameters of SHMAC NoC interconnect

| **Parameter** | **Value** |
|---|---|
| Topology | 2D Mesh |
| Routing | Dimension-ordered (XY) |
| Switching | Store-and-forward |
| Flow control | On/off |
| Router delay | 2 cycles |
| Link delay | 1 cycle |
| Link width | 195 bits |

To speed-up the development of SHMAC prototype, we resort to store-and-forward switching with on/off flow control [8]. However, the switching technique can be replaced in the future if found to cause performance bottlenecks. The effort of this change is minimized, as only the interconnect and the network interfaces have to be updated.

We implement a traditional multi-stage router architecture [8]. All routers contain five ports, for the East, North, West, South and Local connections, respectively. The router model is presented in Fig. 5 and implements two stages: 1) route computation and 2) switch allocation and traversal. The number of stages is minimized to reduce the latency through router. Route computation is done using the XY-routing algorithm. The switch allocation stage performs arbitration for competing inputs (using Round-Robin scheme), and sets the crossbar according to the result of arbitration. A packet which has been granted the access traverses the switch during the same stage.

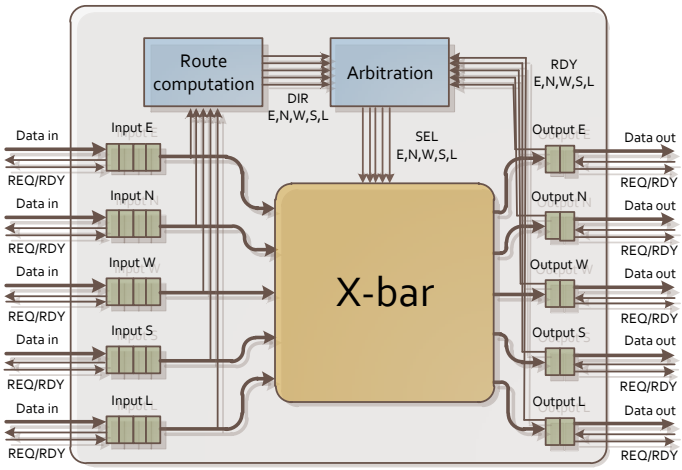


Figure 5: SHMAC Router Architecture

Table 2: Work Package Structure

| Task | Title | WP Responsible |
|---|---|---|
| WP1 | Implementing the SHMAC Infrastructure | Magnus Jahre |
| WP2 | SHMAC System Software | Lasse Natvig |
| WP3 | Exploiting Near- and Sub-threshold Technology in SHMAC | Snorre Aunet |
| WP4 | A System Scenario-based Methodology for SHMAC | Per Gunnar Kjeldsberg |

# 3 Work Packages

In this section, we describe how we intend to realize the SHMAC vision. Figure 6 contains a high-level Gantt chart of the project and the main work packages and the person responsible is shown in Table 2. We have included more work in this plan than what we currently have the resources to complete. These work packages have the status "Future Task" while the work packages that are being worked on are give the status "In progress".

Figure 6 illustrates the SHMAC project's 12 month milestones:

- *Milestone 1 (Q2 2013):* SHMAC 1.0 available and master student graduation
- *Milestone 2 (Q2 2014):* SHMAC internal release and master student graduation
- *Milestone 3 (Q2 2015):* SHMAC 2.0 available and master student graduation

SHMAC is currently in version 1.0 with the last release in September 2013. The SHMAC version numbering system consists of a major number and a minor number. The major number is increased when new functionality is added to SHMAC while the minor number is increased when bugs are fixed.

To ensure efficient collaboration, we will use the SVN and Mercurial version control systems. All papers related to SHMAC should use the dmpub repository[1] while the SHMAC source code will be available in a Mercurial repository[2]. For latex documents, we should have one sentence one one line for easy editing and clean diffs. We use Mercurial for the SHMAC source since it has better support for merging code from different branches.

In addition, the SHMAC project uses Redmine[3] to track bugs and disseminate internal information within the project. It also contains a wiki for interactive project documentation. We encourage all SHMAC users and developers to add useful information to this wiki. Finally, pre-synthesized bit files for the released SHMAC versions are also available in Redmine. We use two different SHMAC-configurations, one with 3 processor cores and one with 8 processor cores. The details regarding each configuration is available in Redmine.

[1] `http://basar.idi.ntnu.no/svn/dmpub`
[2] `ssh://login.idi.ntnu.no//home/felles/card/shmac`
[3] `http://redmine.idi.ntnu.no/projects/shmac`

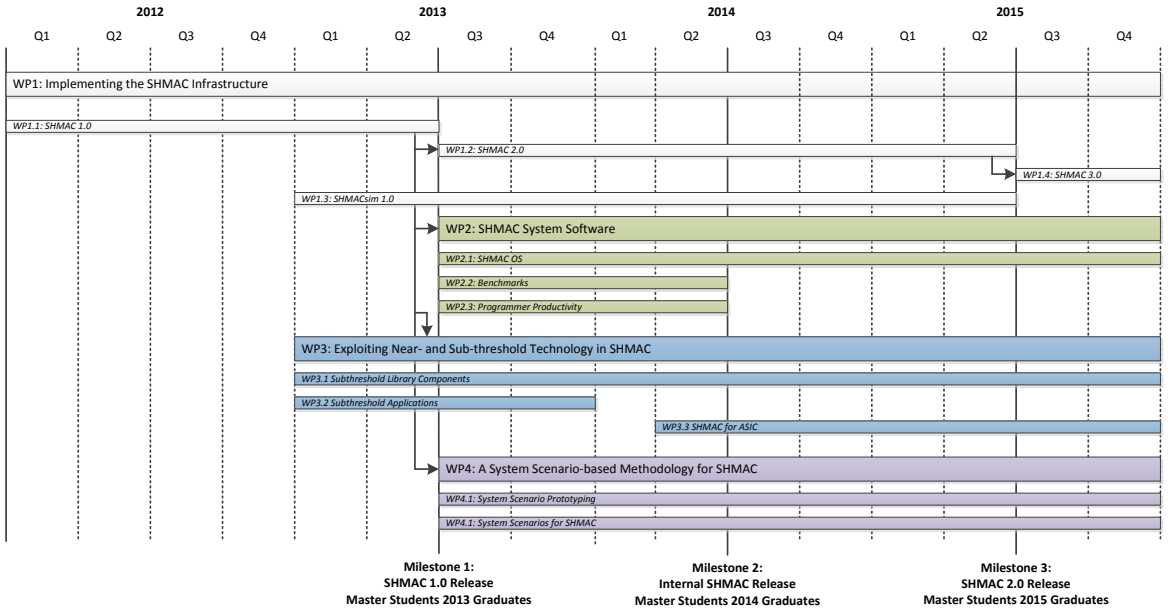


Figure 6: SHMAC Project Schedule

## 3.1 WP1: Implementing the SHMAC Infrastructure

WP1 has two main goals. The first is to build and maintain the SHMAC infrastructure so that it can be used efficiently by researchers working on the other work packages. The second goal is to produce high-quality research results investigating architectural techniques and trade-offs for heterogeneous computing systems.

Table 3 shows the internal work packages of WP1 and the responsible team members. WP1.1 was completed in August/September 2013 and delivered SHMAC 1.0. WP1.2 and WP1.3 develops the SHMACsim simulator and SHMAC 2.0, respectively. The current plan is an internal release of SHMAC 2.0 in Q2 2014 (Milestone 2) and public release in Q2 2015 (Milestone 3). When SHMAC 2.0 is released, we will start working on SHMAC 3.0.

The exact features to include in SHMAC 2.0 depends heavily on the progress of the project as a whole, our ability to attract external funding and the needs of the other WPs. Thus, work packages may be moved from SHMAC 2.0 to SHMAC 3.0 or vice versa.

### 3.1.1 WP1 Research outcomes

- Insight into opportunities and challenges of the Single ISA ambition
- Insight into how accelerators should be designed and included in a heterogeneous computing system

### 3.1.2 WP1 Deliverables

1. Master Thesis: Implementing a Heterogeneous Multi-Core Prototype in an FPGA (WP1.1.1), Complete
2. System: SHMAC Infrastructure 1.0 (WP1.1.3), Complete
3. Master Thesis: SHMACsim - A Cycle-accurate Simulation Infrastructure for the Heterogeneous SHMAC Multi-Core Prototype (WP1.2.1), Complete
4. Project: SimplePipeline - A Processor Core Model for SHMACsim (WP1.2.2), Complete
5. Project (WP1.3.3), In progress
6. Master Thesis (WP1.3.4), In progress

Table 3: WP1 Tasks

| WBS | Task | Resources | Status |
|---|---|---|---|
| **WP1.1** | **SHMAC 1.0** | | |
| *WP1.1.1* | *SHMAC Protype* | *Leif Tore Rusten, Gunnar Inge Sortland* | *Complete* |
| *WP1.1.2* | *SHMAC Initial Architecture* | *Asbjørn Djupdal, Nikita Nikitin* | *Complete* |
| **WP1.2** | **SHMACsim 1.0** | | |
| *WP1.2.1* | *SHMACsim Prototype* | *Yaman Umuroglu* | *Complete* |
| *WP1.2.2* | *Pipelined processor model* | *Anders Akre, Mayeul Marcadella, Sebastian Bøe* | *Complete* |
| *WP1.2.3* | *Integration and Validation* | | *Future Task* |
| **WP1.3** | **SHMAC 2.0** | | |
| *WP1.3.1* | *AMBER support for ARM v3* | *Håkon Furre Amundsen, Joakim Erik Christopher Andersson* | *Complete* |
| *WP1.3.2* | *Porting SHMAC to Versatile Express* | *Asbjørn Djupdal* | *Complete* |
| *WP1.3.3* | *Floating Point Unit support* | *Jakob Dagsland Knutsen* | *In progress* |
| *WP1.3.4* | *A High Performance Tile for SHMAC* | *Anders Akre, Sebastian Bøe* | *In progress* |
| *WP1.3.5* | *Tile with accellerator support* | *Marton Teilgård* | *In progress* |
| *WP1.3.6* | *A Cellular Automata accelerator for SHMAC* | *Einar Johan Trøan Sømåen* | *In progress* |
| *WP1.3.7* | *Configurable Floating Point Unit* | *Audun Indergaard* | *In progress* |
| *WP1.3.8* | *SHMAC 2.0 Integration* | | *Future Task* |
| **WP1.4** | **SHMAC 3.0** | | |
| *WP1.4.1* | *A Vector Tile for SHMAC* | | *Future Task* |
| *WP1.4.2* | *Virtual memory support* | | *Future Task* |
| *WP1.4.3* | *JTAG Inspired Debug/Trace support* | | *Future Task* |

7. Master Thesis (WP1.3.5), In progress
8. Master Thesis (WP1.3.6), In progress
9. Master Thesis (WP1.3.7), In progress
10. System: SHMAC Infrastructure 2.0 (WP1.3.9)

## 3.2 WP2: SHMAC System Software

The aim of WP2 is to facilitate rapid exploration of system software related research topics. To achieve this goal, it is important to identify subsystems that we can reuse across research project. This will reduce the implementation effort per project and improve research project turn-around time.

The SHMAC high-level architecture discussed in Section 2 has significant implications for the system software layer. In particular, the SHMAC architecture assumes:

- SHMAC has no inter-tile cache coherence. Consequently, shared data must be handled explicitly in the software layers.
- SHMAC implements a message passing communication model.
- SHMAC does not implement any protection mechanisms. Consequently, all programs can access all memory locations, and a corrupt program can destroy other processors data structures. On the other hand, maintaining protection does incur some overhead.

Figure 7 illustrates the possible system software layers for SHMAC. We make the distinction between the Operating System (OS) and the runtime system. An OS has a kernel that at least implements scheduling, communication and low level resource allocation services [3]. Runtime systems often implement services that achieve similar

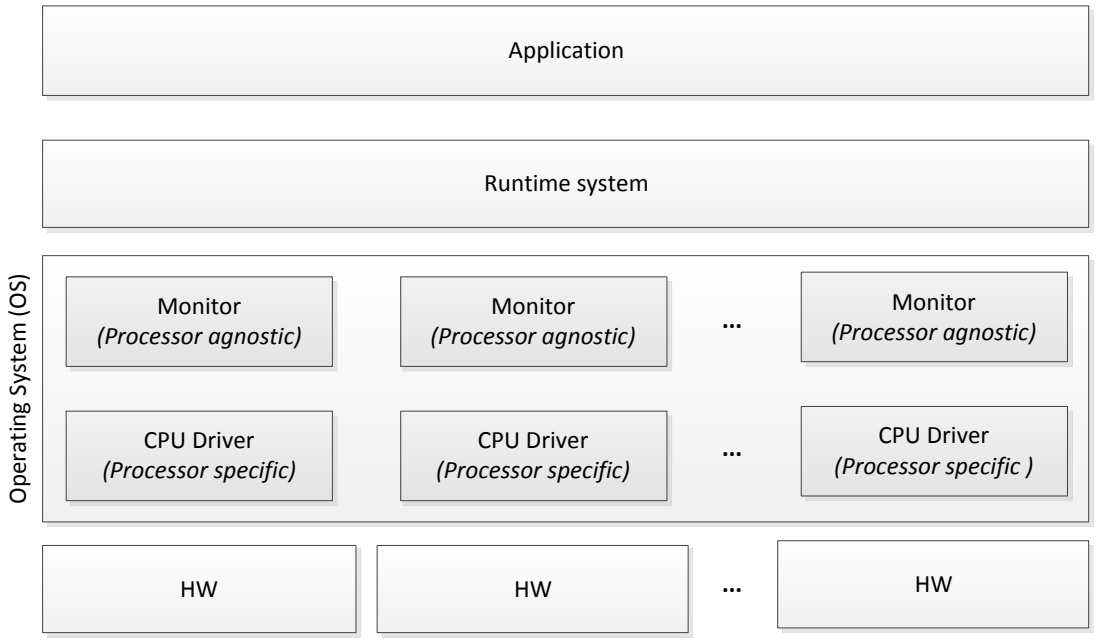


Figure 7: SHMAC System Software Abstraction Layers

goals at a higher abstraction level. Consequently, they commonly build on the services provided by the OS. In addition to the well known operating systems Linux and Windows, research operating systems like Barrelfish [3] and Cory [7] have been proposed for future many-core systems. There is also a significant amount of research into runtime systems for heterogeneous systems like Harmony [10], StarPU [1], OmpSs [11], Myrmics [17], Intel Threading Building Blocks (TBB) [23] and OpenCL [21]. Some of these systems are pure runtime systems while others also require a distinct programming model.

Table 4 shows the subpackages within WP2. WP2.1 aims to port the Barrelfish and Linux operating systems to SHMAC. In WP2.2, the aim is to develop benchmarks for SHMAC. Finally, WP2.3 will make programming SHMAC more efficient by supporting a debugger.

### 3.2.1 WP2 Research outcomes

- Insight into how software should be designed for heterogeneous processors
- Insight into which accelerators are appropriate and how they can be programmed efficiently

### 3.2.2 WP2 Deliverables

1. Project: Porting and Evaluating Barrelfish for SHMAC (WP2.1.1), Complete
2. Project: SHMAC Operating System (WP2.1.2), Complete
3. Master Thesis (WP2.1.4)
4. Master Thesis (WP2.1.5), In progress
5. Project: Benchmarking SHMAC (WP2.2.1), Complete

Table 4: WP2 Tasks

| WBS | Task | Resources | Status |
|---|---|---|---|
| **WP2.1** | **SHMAC OS** | | |
| *WP2.1.1* | *A Distributed OS for SHMAC* | *Benjamin Bjørnseth, Bjørn Christian Seime* | *Complete* |
| *WP2.1.2* | *Linux for SHMAC* | *Håkon Furre Amundsen, Joakim Erik Christopher Andersson* | *Complete* |
| *WP2.1.3* | *Energy Evaluation of SHMAC Sofware* | *Benjamin Bjørnseth* | *Complete* |
| *WP2.1.4* | *A Distributed OS for SHMAC (Master Thesis)* | *Benjamin Bjørnseth* | *Future Task* |
| *WP2.1.5* | *Linux for SHMAC (Master Thesis)* | *Håkon Furre Amundsen, Joakim Erik Christopher Andersson* | *In progress* |
| **WP2.2** | **Benchmarks** | | |
| *WP2.2.1* | *Benchmarking SHMAC* | *Håkon Opsvik Wikene* | *Complete* |
| *WP2.2.2* | *A Prototype Task-Based Parallel Runtime for SHMAC* | *Magnus Walstad* | *Complete* |
| *WP2.2.3* | *Benchmarking SHMAC* | *Håkon Opsvik Wikene* | *In progress* |
| *WP2.2.4* | *Task Based Parallel Programming on the SHMAC Multi-Core Prototype* | *Magnus Walstad* | *In progress* |
| **WP2.3** | **Programmer Productivity** | | |
| *WP2.3.1* | *Prototyping a Debugger for SHMAC* | *Terje Schjelderup* | *Complete* |
| *WP2.3.2* | *Debugger for SHMAC* | *Bjørn Christian Seime* | *In progress* |
| *WP2.3.3* | *A programming environment for SHMAC* | | *Future Task* |

6. Project: Task-Based Programming on SHMAC (WP2.2.2), Complete
7. Master Thesis (WP2.2.3), In progress
8. Master Thesis (WP2.2.4), In progress
9. Project: Debugging Environment for SHMAC (WP2.3.1), Complete
10. Master Thesis (WP2.3.2), In progress

## 3.3 WP3: Exploiting Near-/Sub-threshold Technology in SHMAC

Dependencies between supply voltage, delay, power consumption and energy per computation, for 32-bit addition are shown for a typical 70 nm CMOS technology in Figure 8 [4]. Reducing the supply voltage from 1.0 to about 0.1 V increases the delay of the addition of 3-4 orders of magnitude (1000-10000 times). If the supply voltage is reduced from 1.0 to 0.1 V, the power consumption (W) is reduced about 4 orders of magnitude, as may be seen from Figure 8. The energy per switching [J] has an optimum around 0.3 V, which is a supply voltage close to the absolute value of the threshold voltages of the PMOS and NMOS transistors. For energy consumption the effect of reducing the supply voltage was a maximum gain of about 1 order of magnitude, which is not untypical, but less dramatic than for the delay, which directly affects the maximum clock frequency for a clocked sequential circuit, or the power consumption. Simulations demonstrate the potential of reducing the power supply voltage for CMOS circuitry, which is the dominant IT-platform [4]. Any finite state machine or computing system rely on combinatorial circuits and memory. Measurements on silicon, for a 32-bit RISC processor containing SRAM, implemented in 65 nm CMOS, has proven savings in energy per switching of a factor 11 [16].

A system like in Figure 9 may be implemented as a combination of standard library building blocks and ultra low voltage / low energy building blocks which communicate through voltage level converters. Input and output may link to data converters (A/D and D/A) or other technologies. Ultra low energy state machines, or general digital systems, rely on combinatorial circuits and memory. Combinatorics include for example basic functions

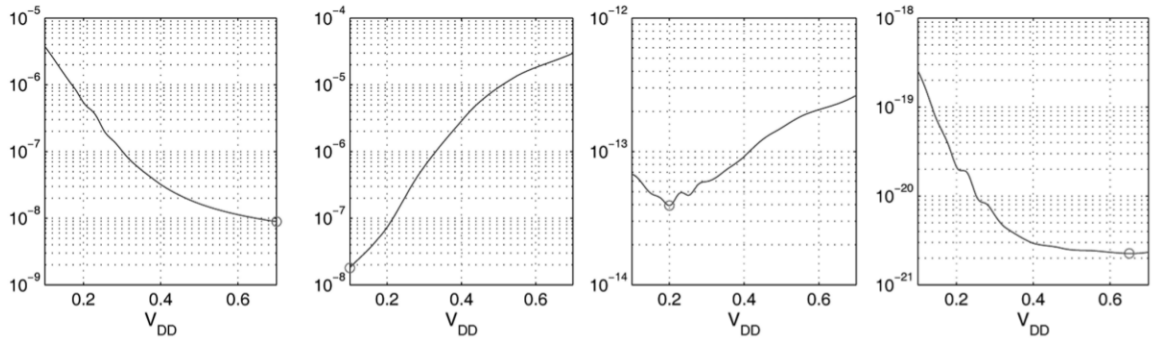


Figure 8: Effects of Varying Supply Voltage in 70 nm CMOS (Reproduced from [4])

like NAND. NOR, EXOR, INVERT etc. Memory could be latches, flip-flops and SRAM, for example. For our hybrid computing systems we will exploit the design space exploring tradeoffs between supply voltage and energy consumption, power consumption and maximum frequency, as such metrics are highly dependent on the supply voltage of choice, as indicated in Figure 8.

Table 5 shows the subtasks in WP3. These tasks all contribute towards efficiently exploiting low supply voltage technologies in SHMAC as exemplified in Figure 9. The aim of WP3.1 is to build a cell library which makes it possible to efficiently synthesize HDL code with a mixture of regular and low voltage technologies. WP3.2 is aimed at gathering knowledge on the practical aspects of applying near- and subthreshold technology to real world problems. Finally, WP3.3 links basic research isn WP3.1 and WP3.2 to the SHMAC HDL from WP1. This task is motivated by the fact that an ASIC realization is needed to accurately measure the efficiency of near- and subthreshold technologies. WP3.3 is scheduled to start in Q3 2014.

### 3.3.1 WP3 Research outcomes

- Insight into how near- and sub-threshold techniques can be applied at the micro- and macro-architecture abstraction levels

### 3.3.2 WP3 Deliverables

1. Master Thesis: Trade-offs between Performance and Robustness for Ultra Low Power/Low Energy Subthreshold D flip-flops in 65nm CMOS (WP3.1.1), Complete
2. Project: Ultra-Low Voltage Flipflops Standard Cell Library (WP3.1.2), Complete
3. Project: Ultra-Low Voltage SRAM Cell (WP3.1.3), Complete
4. Project: Combinatorial CMOS Cell library supporting dynamic supply voltages (WP3.1.4), Complete
5. Master Thesis (WP3.1.5), In progress
6. Master Thesis (WP3.1.6), In progress
7. Master Thesis (WP3.1.7), In progress
8. PhD Thesis (WP3.1.8), In progress
9. Master Thesis: Low Energy Implementation of Robust Digital Arithmetic in Sub/Near-Threshold Nanoscale CMOS - for ultrasound beamforming (WP3.2.1), Complete
10. Master Thesis: Subthreshold Real-Time Counter (WP3.2.2), Complete
11. Project: Design of Parallel correlation modules in Subthreshold to be used in CDMA Systems (WP3.2.3), Complete

Table 5: WP3 Tasks

| WBS | Task | Resources | Status |
|---|---|---|---|
| **WP3.1** | **Subthreshold library components** | | |
| *WP3.1.1* | *Performance/Robustness Trade-offs in D Flip-Flops* | *Magne Værnes* | *Complete* |
| *WP3.1.2* | *Ultra-Low Voltage Flip-Flop Library (with Atmel)* | *Joakim Haram* | *Complete* |
| *WP3.1.3* | *Ultra-Low Voltage SRAM Cell (with Atmel)* | *Ole Samstad Kjøbli* | *Complete* |
| *WP3.1.4* | *Combinatorial CMOS with dynamic supply voltage (with Atmel)* | *Hallstein Skjølsvik* | *Complete* |
| *WP3.1.5* | *Subthreshold Library Cell Development (with Atmel)* | *Joakim Haram* | *In progress* |
| *WP3.1.6* | *Subthreshold Library Cell Development (with Silicon Labs)* | *Hallstein Skjølsvik* | *In progress* |
| *WP3.1.7* | *Subthreshold Library Cell Development (with Atmel)* | *Glenn Andre Johnsen* | *In progress* |
| *WP3.1.8* | *Developing a near-/-sub threshold cell library* | *Ali Asghar Vatanjou* | *In progress* |
| **WP3.2** | **Subthreshold Applications** | | |
| *WP3.2.1* | *Robust Sub/Near-threshold Arithmetic in Ultrasound* | *Lars-Frode Schjolden* | *Complete* |
| *WP3.2.2* | *Subthreshold Real-Time Counter* | *Jonathan Edvard Bjerkedok* | *Complete* |
| *WP3.2.3* | *Subthreshold Parallel Correlation Modules for CDMA (with Q-Free)* | *Glenn Andre Johnsen* | *Complete* |
| **WP3.3** | **SHMAC for ASIC** | | |
| *WP3.3.1* | *Synthesizing SHMAC for ASIC* | *Benjamin Bjørnseth* | *On Schedule* |
| *WP3.3.2* | *Developing energy models* | *Benjamin Bjørnseth* | *Future Task* |
| *WP3.3.3* | *A CPU microarchitecture with near-/sub- threshold support* | | *Future Task* |
| *WP3.3.4* | *Prototype* | | *Future Task* |

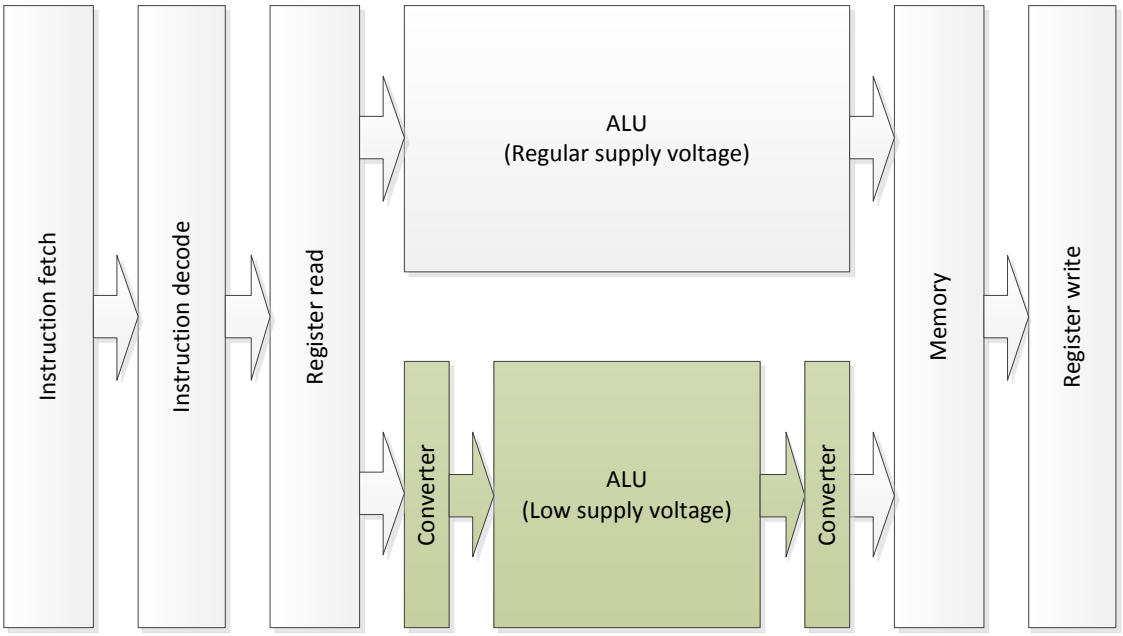


Figure 9: Processor Pipeline with Low Supply Voltage Support

12. System: Energy models for SHMAC (WP3.3.2)
13. System: A microarchitecture implementation capable of leveraging near-/sub-threshold technology (WP3.3.3)
14. System: An ASIC realization of SHMAC (WP3.3.4)

## 3.4 WP4: A System Scenario-based Methodology for SHMAC

For heterogeneous multiprocessor systems running dynamic applications, optimized mapping and scheduling of tasks on processing units is essential to achieve energy efficiency. Examples of dynamic applications can be found in domains such as biomedical technology, communication, and multi-media. The dynamic behavior should be exploited to reconfigure the platform to the current needs of, e.g., performance and memory space. This way the overhead inflicted by always assuming worst-case requirements is avoided. Examples of reconfiguration ”knobs” are dynamic voltage and frequency scaling (DVFS), mapping of tasks on processing units with different performance/energy profiles, turning memories into low power retention mode if data is not needed for a time period, and turning processing units and memories off through power gating if they are not used at all for a time period.

The dynamic behavior can be exploited through using a system scenario design approach [13]. A system scenario is a configuration of the system that combines similar run-time situations (RTSs). An RTS consists of a running instance of a task and its corresponding cost (e.g., energy consumption) and one complete run of the application on the target platform represents a sequence of RTSs [14]. The system is configured to meet the cost requirements of an RTS by choosing the appropriate system scenario, which is the one that satisfies the requirements using minimal energy. The system scenario implementation is divided into a design-time and run-time part to achieve optimized solutions without having a too costly adaptation process at run-time [18]. At design-time the application to be implemented is profiled with an extensive set of input data. A gray-box model is generated in which static parts of the code are hidden from the designer while the parts that generate dynamic behavior are available for

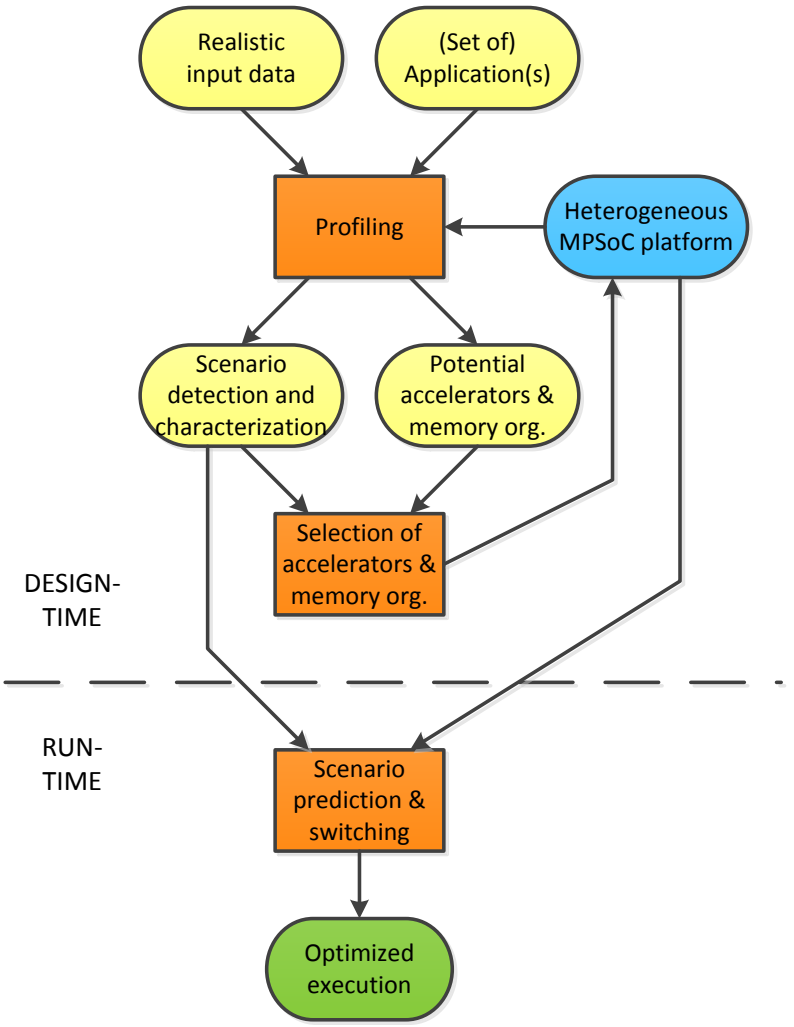


Figure 10: System scenario design approach with platform adaption

optimization. Control- and data-variables that decide the dynamism are also detected during the profiling. The next step is a clustering of the RTSs into a limited number of scenarios with similar characteristics. Based on the most influential control- and data-variables, a prediction and switching mechanism is generated that at run-time can be used to predict what the next scenario should be and switch accordingly.

If multiple tasks are competing for he same resources, each RTS, and hence each scenario, will consist of a set of Pareto optimal configurations. These are found through an exploration of alternative solutions, i.e., knob settings, for each RTS. All solutions that are optimal for one optimization criteria are kept. If we, for instance, are simultaneously optimizing for energy and speed, the most energy efficient solution for each speed level is kept as part of a two-dimensional Pareto curve. At run-time, a sequence of RTS Pareto points are selected for each of the competing tasks so that the global energy consumption is minimized while at the same time all deadlines are met. Such an approach can also be used if we need to have, e.g., an adaptive quality of service. The configuration can then be changed to give a lower quality of service when the remaining battery life time is below a certain limit.

Compared with use case scenario approaches, in which scenarios are generated based on a user's behavior, the system scenario methodology focuses on the behavior of the system to generate scenarios and can, therefore, fully exploit the detailed platform mapping information.

In situations where we have a limited number of applications, and hence different types of tasks, the computing platform can be fine-tuned to the set of scenarios at hand. At design-time a set of accelerators can be selected

Table 6: WP4 Tasks

| WBS | Task | Resources | Status |
|---|---|---|---|
| **WP4.1** | **System Scenario Prototyping** | | |
| *WP4.1.1* | *Basic block accelerators* | *Marton Teilgård, Sunniva Berg* | *Complete* |
| *WP4.1.2* | *Accelerator integration* | *Yves Bernard* | *Complete* |
| *WP4.1.3* | *Mapping an Epileptic Seizure Prediction Algorithm to SHMAC* | *Sunniva Berg* | *In progress* |
| *WP4.1.4* | *Optimization of memory organization for system scenario design* | *Iason Filippopoulos* | *In progress* |
| **WP4.2** | **System Scenarios for SHMAC** | | |
| *WP4.2.1* | *Implementation of prediction and switching on the SHMAC architecture* | *Yahya Yassin* | *In progress* |
| *WP4.2.2* | *Mapping of relevant dynamic applications on the SHMAC architecture with scenarios* | | *Future Task* |
| *WP4.2.3* | *Complete system scenario design approach with SHMAC architecture adaptation* | | *Future Task* |

for inclusion in the platform. At run-time the platform can be adapted, e.g., through powering accelerators and computing resources on and off and through reconfiguration of communication and memory hierarchies. The SHMAC-platform is very well suited for exploration of the system scenario design approach and in particular experiments with platform adaption and reconfiguration. Figure 10 depicts the envisioned design approach.

Table 6 shows the subpackages within WP4.

### 3.4.1 WP4 Research outcomes

- Insight into how tasks should be scheduled and mapped the cores of heterogeneous processors at design-time and run-time
- Insight into how the architecture can be optimized with accelerators for system scenario exploitation

### 3.4.2 WP4 Deliverables

1. Project: Evaluation of basic block accelerators for use on the SHMAC platform (WP4.1.1), Complete
2. Project: Alternative approaches for integration of accelerators in a heterogeneous platform (WP4.1.2), Complete
3. Master Thesis (WP4.1.3), In progress
4. PhD Thesis (WP4.1.4), In progress
5. System: A System Scenario Methodology for SHMAC (WP4.2.4)

Table 7: SHMAC Risks

| ID | Risk | Probability | Impact | Total |
|---|---|---|---|---|
| R1 | SHMAC implementation delayed which again delays other tasks | 2 | 5 | 10 |
| R2 | SHMAC implementation complexity overwhelming | 3 | 3 | 9 |
| R3 | Lack of funding for prototyping activities | 3 | 3 | 9 |
| R4 | Tasks to difficult for researcher | 2 | 4 | 8 |
| R5 | Time leak for faculty | 4 | 2 | 8 |
| R6 | Lack of funding for research activities | 2 | 4 | 8 |
| R7 | Intellectual Property problems: infringing on patents or licensed products | 1 | 5 | 5 |
| R8 | Cannot recruit enough master students | 3 | 1 | 3 |

Table 8: Risk (R) Mitigation through Prevention (P) and Correction (C)

| | R1 | R2 | R3 | R4 | R5 | R6 | R7 | R8 |
|---|---|---|---|---|---|---|---|---|
| P1 | X | X | | X | | | | |
| P2 | X | X | | X | | | | |
| P3 | X | X | | X | | | | |
| P4 | | | X | | | X | | |
| P5 | X | X | | X | X | | | |
| P6 | | | | | | | X | |
| P7 | | | | | | | | X |
| C1 | X | X | | X | | | | |

# 4 Risks

Table 7 summarizes and scores the most important risk factors in the SHMAC project. The aim of including this section in the plan is to develop strategies that avoid most of the problems can occur and to be able to efficiently mitigate the problems that do occur.

The SHMAC project implements the following preventive measures:

1. Implement smaller simpler systems to gain experience before taking on the big challenges
2. Build on scientific best practices and leverage the work of other when it is appropriate
3. Implement good hiring procedures that ensure that we employ the best possible people
4. Apply for funding regularly and try to be creative in seeking out funding opportunities
5. Faculty strictly prioritize their time and delegate where it is possible and efficient
6. The SHMAC project management understand the license agreements of all 3rd party technologies
7. Faculty defines clear and motivating master thesis topics early

The SHMAC project implements the following corrective measures:

1. If the problem at hand is to difficult, simplify the problem and solve the simple problem first

Table 8 shows how the preventive and corrective measures reduce the probability and impact of the risks identified in Table 7.

# 5 Conclusion and Future Directions

This report has described how we plan to reach the research goals of the SHMAC project. In addition to the main line SHMAC development (WP1), we plan to investigate system software issues for heterogeneous architectures (WP2) as well as how near- or sub-threshold technology (WP3) and system scenario-based optimization (WP4) can be applied to the SHMAC infrastructure. Developments in WP1 will be made available to the other work packages every 12 months.

We have discussed a few ideas that have not been included in this plan but that might be interesting to investigate in a longer term:

- We can consider to add support for hardware cache coherence in SHMAC. With coherent caches, software can treat memory as globally shared which may simplify development. On the other hand, implementing cache coherence is challenging and the overhead of its implementation grows with the number of cores. Consequently, we leave cache coherence as further work.
- Modern computing systems contain a collection of power and energy saving techniques like power-gating, clock-gating, DVFS, etc. The payoff from these techniques is limited in an FPGA realization since you can implement the largest system your application can make use of and turn the rest of the FPGA resources off. On the other hand, these techniques are interesting for an ASIC realization. Unfortunately, they increase the verification effort significantly. Consequently, we will not implement these techniques for the first generations of ASIC SHMAC but may consider implementing them in later versions.

# References


[1] C. Augonnet, J. Clet-Ortega, S. Thibault, and R. Namyst. Data-aware task scheduling on multi-accelerator based platforms. In *Parallel and Distributed Systems (ICPADS), 2010 IEEE 16th International Conference on*, pages 291–298, 2010.

[2] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel. Scratchpad Memory: Design Alternative for Cache On-chip Memory in Embedded Systems. In *Proceedings of the Tenth International Symposium on Hardware/Software Codesign*, CODES '02, pages 73–78, 2002.

[3] A. Baumann, P. Barham, P.-E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania. The Multikernel: a new OS Architecture for Scalable Multicore Systems. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 29–44, 2009.

[4] V. Beiu, S. Aunet, J. Nyathi, R. Rydberg, and W. Ibrahim. Serial Addition: Locally Connected Architectures. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 54(11):2564–2579, 2007.

[5] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, M. Mattina, C.-C. Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook. TILE64 - Processor: A 64-Core SoC with Mesh Interconnect. In *ISSCC 2008: Solid-State Circuits Conference*, 2008.

[6] S. Borkar and A. A. Chien. The Future of Microprocessors. *Commun. ACM*, 54(5):67–77, 2011.

[7] S. Boyd-Wickizer, H. Chen, R. Chen, Y. Mao, F. Kaashoek, R. Morris, A. Pesterev, L. Stein, M. Wu, Y. Dai, Y. Zhang, and Z. Zhang. Corey: an Operating System for Many Cores. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, pages 43–57, 2008.

[8] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., 2003.

[9] R. Dennard, F. Gaensslen, V. Rideout, E. Bassous, and A. LeBlanc. Design of ion-implanted MOSFET's with very small physical dimensions. *Solid-State Circuits, IEEE Journal of*, 9(5):256–268, 1974.

[10] G. F. Diamos and S. Yalamanchili. Harmony: an Execution Model and Runtime for Heterogeneous Many Core Systems. In *Proceedings of the 17th international symposium on High performance distributed computing*, HPDC '08, pages 197–200, 2008.

[11] A. Duran, E. Ayguadé, R. M. Badia, J. Labarta, L. Martinell, X. Martorell, and J. Planas. Ompss: A proposal for programming heterogeneous multi-core architectures. *Parallel Processing Letters*, 21:173–193, 2011.

[12] H. Esmaeilzadeh, E. Blem, R. St.Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pages 365–376, 2011.

[13] S. V. Gheorghita, M. Palkovic, J. Hamers, A. Vandecappelle, S. Mamagkakis, T. Basten, L. Eeckhout, H. Corporaal, F. Catthoor, F. Vandeputte, and K. D. Bosschere. System-scenario-based Design of Dynamic Embedded Systems. *ACM Trans. Des. Autom. Electron. Syst.*, 14(1):3:1–3:45, Jan. 2009.

[14] E. Hammari, F. Catthoor, J. Huisken, and P. G. Kjeldsberg. Application of medium-grain multiprocessor mapping methodology to epileptic seizure predictor. In *NORCHIP, 2010*, pages 1 –6, nov. 2010. doi: 10.1109/NORCHIP.2010.5669489.

[15] J. L. Hennessy and D. A. Patterson. *Computer Architecture - A Quantitative Approach, Fourth Edition*. Morgan Kaufmann Publishers, 2007.

[16] S. Lutkemeier, T. Jungeblut, H. Berge, S. Aunet, M. Porrmann, and U. Ruckert. A 65 nm 32 b Subthreshold Processor With 9T Multi-Vt SRAM and Adaptive Supply Voltage Control. *Solid-State Circuits, IEEE Journal of*, 48(1):8–19, 2013.

[17] S. Lyberis, P. Pratikakis, D. S. Nikolopoulos, M. Schulz, T. Gamblin, and B. R. de Supinski. The Myrmics Memory Allocator: Hierarchical, Message-passing Allocation for Global Address Spaces. In *Proceedings of the 2012 international symposium on Memory Management*, pages 15–24, 2012.

[18] Z. Ma, P. Marchal, D. P. Scarpazza, P. Yang, C. Wong, J. I. Gómez, S. Himpe, C. Ykman-Couvreur, and F. Catthoor. *Systematic Methodology for Real-Time Cost-Effective Mapping of Dynamic Concurrent Task-Based Systems on Heterogenous Platforms*. Springer Publishing Company, Incorporated, 1st edition, 2007. ISBN 1402063288, 9781402063282.

[19] K. Olukotun and L. Hammond. The Future of Microprocessors. *Queue*, 3(7):26–29, 2005.

[20] L. T. Rusten and G. I. Sortland. Implementing a Heterogeneous Multi-Core Prototype in an FPGA. Master's thesis, NTNU, 2013.

[21] J. Stone, D. Gohara, and G. Shi. OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. *Computing in Science Engineering*, 12(3):66–73, 2010.

[22] Y. Taur and E. Nowak. CMOS devices below 0.1 mu;m: how high will performance go? In *Electron Devices Meeting, 1997. IEDM '97. Technical Digest., International*, pages 215–218, 1997.

[23] TBB. Intel Threading Building Blocks. `http://software.intel.com/sites/products/documentation/hpc/tbb/getting_started.pdf`, Retrieved March 2013.

[24] M. Zhang and K. Asanovic. Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors. In *Proceedings of the 32nd annual international symposium on Computer Architecture*, ISCA '05, 2005.