## Change log

<table>
<thead>
<tr>
<th>VERSION</th>
<th>DESCRIPTION OF CHANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>V0.1</td>
<td>First draft USIEGEN</td>
</tr>
<tr>
<td>V0.2</td>
<td>Second Draft ALL</td>
</tr>
<tr>
<td>V0.3</td>
<td>Consolidated bay USI</td>
</tr>
<tr>
<td>V0.4</td>
<td>Reviewed by KTH</td>
</tr>
<tr>
<td>V0.5</td>
<td>Final version considering the comments and with Word format adjustments.</td>
</tr>
<tr>
<td>V1.0</td>
<td>Submitted version</td>
</tr>
</tbody>
</table>
# Table of contents

1. EXECUTIVE SUMMARY ............................................................................. 4

2. INTRODUCTION......................................................................................... 5

3. OBJECTIVES............................................................................................... 7
   3.1. SAFETY .................................................................................................. 7
   3.3. SECURITY ............................................................................................... 8
   3.4. POWER EFFICIENCY ............................................................................. 8

4. SAFEPOWER ARCHITECTURE ................................................................. 10
   4.1. APPLICATION MODEL ........................................................................ 10
   4.2. PHYSICAL PLATFORM .......................................................................... 10
   4.3. SCHEDULE ............................................................................................ 13
   4.4. META-SCHEDULER ............................................................................ 14
   4.5. ADAPTIVE ARCHITECTURE ............................................................... 16

5. LOCAL SCOPE ......................................................................................... 20
   5.1. EXECUTION SERVICES ......................................................................... 20
   5.2. LOCAL LOW POWER OPTIMIZATION .............................................. 22

6. GLOBAL SCOPE ...................................................................................... 26
   6.1. ADAPTATION ARCHITECTURE .......................................................... 26
   6.2. COMMUNICATION SERVICES ............................................................ 27
   6.3. GLOBAL CONTEXT ESTABLISHMENT SERVICES .................................. 31
   6.4. DECISION MAKING SERVICES ............................................................ 34

7. MONITORING SERVICES .......................................................................... 37
   7.1. MONITORS AND MONITORING SERVICES ....................................... 37
   7.2. MONITORS SERVICES - INTERFACES TO NETWORK ....................... 38
   7.3. MONITORING SERVICES ON THE ZYNQ SOC ................................. 39

8. LITERATURE .............................................................................................. 46
EXECUTIVE SUMMARY

This document describes the generic architecture which will be implemented in the SAFEPOWER project.
INTRODUCTION

Techniques for power and energy efficiency are increasingly attractive for industry. Power and energy efficiency decreases energy and cooling costs, avoids premature wear-out and failures due to high temperatures and extends the life-time of battery-operated devices. In addition, power and energy efficiency is part of the safety arguments in some application domains (e.g., wireless sensors in safety-critical monitoring applications).

Another important trend is the shift to mixed-criticality systems, which has a significant impact on the system design. Mixed criticality means that applications with varying criticalities run simultaneously on the same system, sharing the provided computational and communication resources. By letting applications of varying criticalities share the same resources instead of providing dedicated separated systems for them, economic and technical advantages are attained. The ensuing systems need less energy, are easier to produce and take up less space and weight in the final industrial product. These attributes have therefore led to great popularity of MCS in today’s industrial systems. But even though MCSes promise improvements regarding system size, cost, power and reliability, these systems face new challenges. Since different criticalities access the same resources and use the same communication medium, the system behaviour must be strictly regulated to make sure that the applications do not interfere with each other, while still allowing them to interact.

Especially in safety-critical control and monitoring applications like in avionics or railways the system must provide precise timing guarantees to ensure that the system behaves in a safe way and no lives are endangered. These guarantees include the task timing, the communication and the resource access.

In order to ensure these guarantees are met, many safety-critical applications employ a precomputed temporal behaviour defined in a schedule. This schedule dictates when which task is allowed to be executed and when messages may be sent.

Such a predictable behaviour is desirable as the system behaviour can be optimized at design time to guarantee timeliness. But since the real runtime of tasks is not known before the actual execution, these schedules are computed based on an assumed Worst Case Execution Time (WCET). This WCET is the maximum time duration that an event, e.g. task or message transmission, needs to be executed. To make sure, that high criticality task finish on-time with a high probability, their WCETs are computed on more pessimistic grounds as compared to low criticality tasks. These assumed WCETs may lead to an unoptimized usage of the system. In average conditions, the tasks will be executed faster than expected, leading to idle times.

In applications, these idle times can be used by dynamic systems to improve the system’s energy usage. Various power saving mechanisms can exploit the idle time by executing tasks at lower frequencies. These dynamic changes are in general not suitable for certifiable systems, because dynamic changes can introduce timing uncertainties.
Additionally, at runtime the environmental changes like the increase in ambient temperature can lead to a performance decrease. Enabling the system to react to such changes can improve its lifetime and therefore reduce the cost for maintenance.

The goal and challenges of SAFEPOWER are to introduce adaptive services in a mixed criticality architecture in order to allow the system to save energy at runtime. For that, the architecture:

1. provides energy/power efficiency for the mixed-criticality system without adverse effects on certifiability (i.e., energy/power efficiency to improve the value of the mixed-criticality system without compromising temporal predictability and fault containment)
2. provides energy/power efficiency for applications where energy/power constraints are part of the safety arguments

This document will describe the baseline architecture and the additional services needed to realize the SAFEPOWER goals. First, the baseline application model and physical platform model will be introduced in chapter three. Additionally, chapter three is providing an overview over the applied scheduling methodologies. Chapter four will focus on new adaptive services and how they will be integrated in the baseline architecture. Chapter five and six will describe the fault tolerance and security of the architecture. And in chapter seven, the industrial partners will trace how the new adaptive services will be of use in the corresponding domains.
1. OBJECTIVES

Based on the goals defined for the SAFEPOWER architecture this section will highlight which particular objectives will be addressed with the architecture.

Safety

There exists no single model for building systems that interact with a physical environment based on many different and, partially, contradicting requirements in the field of computer and communication science. For designing a system, a tradeoff between predictability versus efficiency and flexibility versus fault containment strategies must be considered [1]. Hard real-time systems must be designed according to the resource adequacy policy by providing sufficient computing resources to handle the specified worst-case load and fault scenarios.

Furthermore, the SAFEPOWER architecture supports different timing models: time-triggered and event-triggered control.

- **Time-triggered control**: In time-triggered architectures, activities are triggered by the progression of global time. In such systems, time-triggered messages must be sent over the network at predefined instances and take precedence over all other message types. The occurrence, temporal delay and precision of time-triggered messages are prior known and guaranteed. This type of messages has minimum delay on the network and low jitter. The time-triggered approach is generally preferred for safety-critical systems [2].

- **Event-triggered control**: System activities that are initiated by events in the environment or the computer system are called event-triggered. An activity can be sending of a message or starting computational activities. One can distinguish between two types of event-triggered messages: rate-constrained and best-effort messages.

  The main difference between these two types of messages is that rate-constrained communication guarantees a sufficient bandwidth allocation for each transmission with defined limits for delays and temporal deviations while in the best effort transmission of messages there is no guarantee whether or when the message will arrive at the destination.

Time-Triggered control has a significant role in safety systems. In this case schedules are used to ensure that the system will never reach an unpredicted and unsafe state. These schedules confine the system within certifiable bound but make them rigid and unable to adapt at runtime. In the SAFEPOWER time-triggered approach play a significant role to ensure safety in safety critical systems.
1.1.1. Safety Concept

The architectural approach of SAFEPOWER, both the safety approach and power management services, is being analyzed in a Safety Concept (D2.4 and D2.5) to justify and evaluate the technical compliance of those elements with respect to generic and domain specific safety standards. This safety-concept document is built based on a specific instance of the SAFEPOWER architecture defined in a Railway Use Case, in which a real application is defined and illustrated.

The main objective of the Safety Concept is to go through the cross-domain reference architecture defined in SAFEPOWER for low-power and mixed-criticality systems, and assess its technical compliance with relevant safety certification standards, such as IEC61508 and EN5012x. This will provide sound argumentation and feedback for the safety dimension of the architecture. The Safety Concept assessment is performed by an independent certification authority who approves the solution according to the mentioned standards.

Security

Nowadays, when it comes to safety it’s necessary to consider also the security aspects of the system, in order to avoid dangerous situations caused by malicious access or operation of the system.

SAFEPOWER consists of a generic platform and, as a consequence, it will also define generic security services that allow building solutions to achieve different level of protection depending on specific needs. These security mechanisms are based on the following basic security concepts:

- User management (authentication, authorization, non-repudiation)
- Privacy (encryption)
- Data integrity

SAFEPOWER architecture provides the means to implement different solutions as a combination of hardware and software resources. As part of the Safety Concept and based again on the Railway Use Case, a Security Analysis is being carried out to identify the security requirements that are necessary to achieve a certain level of security in accordance to the resources that need to be protected in that specific case.

This analysis is focused on the security aspects that can have an impact on the safety of the system, i.e. defining those mechanisms of protection that assure integrity of sensitive data and/or detection of data corruption, to ensure that a safe state can always be reached. This analysis will identify potentially relevant implementations of security services for mixed-criticality low-power from which a sub-set could be also implemented.

Power efficiency

At present and for a foreseeable future, power is a resource that has to be shared among many different applications with different criticality levels. Power-management techniques
have a direct influence on improvement of power efficiency in mixed criticality systems. In particular, the combination of DVFS (Dynamic Voltage and Frequency Scaling) and rescheduling techniques can lead to a significant reduction in power. From a variety of analyzed low power techniques within WP1 an applicable subset of low power methods was chosen to be integrated as services within the SAFPOWER architecture. The following methods will be incorporated in the architecture:

1.1.2. Clock gating

Clock gating is a technique for reducing dynamic power by shutting down the elements in the circuit during the time that they do not provide any services to the whole system. This is technically achieved by adding a dedicated circuitry to control the clocks. This unit disables a subset of flip-flops and the subsequent combinatorial logic connected to them to prevent them from unnecessary state switching and to eliminate the switching power, thereby reducing the dynamic power consumption. However, the leakage power cannot be eliminated by this approach.

1.1.3. DVFS

Dynamic Voltage and Frequency Scaling (DVFS) is a solution for decreasing power consumption where the clock frequency of a processor is decreased to allow a corresponding reduction in the supply voltage. DVFS leads to a significant reduction in the required energy for computation, particularly for memory-bound workloads [3]. Distributed DVFS enables power-efficiency, while taking into account real-time transactions with different timing models and temporal constraints.

1.1.4. Rescheduling

To apply the power efficiency techniques, we will implement a static system with quasi-dynamic rescheduling to simulate a dynamic reaction to the system status. These services will change the scheduling and allocation of communication and computational activities at runtime in order to exploit the potential for power and energy efficiency, while at the same time considering the requirements of certification, real-time constraints and time/space partitioning. In particular, solutions for the dynamic reconfiguration of time-triggered execution and communication systems will be provided. Branching points in time-triggered schedule will be analyzed and communication systems will be provided. Branching points in time-triggered schedules will be analyzed and optimized at design time, thereby obtaining temporal guarantees and maintaining the benefits of time-triggered control for Critical Real-Time Embedded Systems such as temporal predictability, implicit synchronization of resource accesses and service execution. Low-power scheduling services will dynamically reconfigure and monitor the communication schedules of time-triggered on-chip networks.
SAFEPOWER ARCHITECTURE

Application Model

A MCS (Mixed-Criticality System) is defined in [1] as a finite number of components $K_i$ which themselves harbor a finite number of tasks $\tau_i$. Each component is assigned a criticality $L_i$, which will be inherited to all associated tasks $\tau \in K_i$. Each task is defined by a Set $(T_i, D_i, C_i, L_i)$ where $T_i$ is the release Time, $D_i$ is the deadline, $C_i$ is the computation time (worst-case execution time assumption), $L_i$ is the task’s criticality.

Many safety critical applications are hard real-time control applications. In those, the achievement of control stability and safety depends on delivering the correct result in bounded time. The latest instant at which the result must be present is called the deadline. A deadline miss represents a failure with the potential consequences of an incorrect result. Even in the case of peak loads or faults, a hard real-time system must guarantee a response. That means timing and resource analysis must always consider the worst-case behaviour of the system. The analysis can be simplified, if the hardware platform ensures determinism and temporal independence of safety-critical components from the non-critical ones. Then, each component has predefined instants when it executes and communicates.

The components in an MCS do not only differ in their criticality, they are also heterogeneous regarding timing (hard, firm, soft or no real-time), hardware requirements (e.g. performance or certifiability) and models of computation. Hard real-time means that all executions must finish in time to prevent failures in a safety-critical application. On the other hand, soft real-time is related to non-safety-critical applications where deadlines might be missed occasionally. If the result has no use after a deadline miss, the timing is called firm.

1.2. Physical Platform

The system model consists of a tile based architecture which is connected via a network on chip. We perform applications of mixed criticality on the tiles which are used in a real-time system. Therefore our architecture is chosen to enable strict temporal and spatial separation to ensure that no low criticality application can impact or delay a higher critical one or access its dedicated resources. The system must also ensure that all tasks can be performed within their associated deadlines. Real-time systems work with time sensitive data that needs to be evaluated and acted on directly, e.g. sensor data. Especially in safety-critical systems missing these deadlines can cost human lives, e.g. if the altitude measurement in flight control systems fails. Figure 1 shows how each tile is managed by a hypervisor running bare-metal on the tile’s hardware and allowing for arbitrary amounts of partitions. The applications on these partitions can be of varying criticality though to keep the spatial separation only the same criticality can be executed in a single partition. The hypervisor assigns hardware resources of its tile to its managed partitions, ensuring the timing and spatial restriction for the mixed criticality system. It also manages the partitions by scheduling their execution by the corresponding deadlines to provide the real-time functionality. Since the partitions do not directly access the hardware on their own, the messages they send are also relayed by the
hypervisor. This prevents message flooding to the NoC and the applications of other tiles are secured from malicious behaviour.

The architecture is applicable to systems which are connected via a bus as well as to systems using a network on chip. In the network on chip version each tile is connected via a network interface (NI) to a time-triggered NoC. The time triggered behaviour is executed based on an a priori computed communication schedule which commands the message injection times. The schedule is enforced by the NIs which queue the messages from the core and inject them into the network based on the precompiled communication schedule. This schedule ensures that no packets collide during the transmission and guarantees timing bound for the message traversal. For this, it does not only provide the message injection but also the path the message is routed on. In addition, the NI contains a scheduler in the packetization and flitization units which compares the current time to the schedule and initiates the message injection.

![Figure 1: Physical platform model showing the tile based architecture with the NoC and Interfaces.](image)

The much smaller softcores are not organized by a hypervisor, as they are meant to be used bare metal. To create a transparent view on the system for applications, the softcores will implement a light version of the XtratuM application interface. This way the applications can be placed on
all the different systems cores without regards for their type, e.g. softcore or processor, provided that the cores. The network on chip is time triggered to ensure strict temporal and spatial partitioning by timing the message transmission and avoiding the shared usage of links by two messages at once. A second advantage of the pre-planned execution and communication is that the system can provide timing guarantees a priori based on the scheduled upper bound for events. The architecture will support hardware and software status monitors. The former will be implemented by dedicated hardware elements and the latter software status will be provided by the hypervisor (or the baseline of the core application if implemented in bare-metal). The software status information will include the application performance in regards to its scheduled behaviour (e.g. slack times, faults etc.).

1.2.1. Network on Chip

In bus-based interconnection architectures, the growing number of IP modules in Systems-on-Chips (SoCs) may prevent these systems to meet the demands of high performance communication infrastructures required by many applications. Despite the fact that many systems with intensive parallel communication buses offer significant benefits with respect to predictability and fault containment they have strict requirements with respect to bandwidth, latency, and power consumption. A Network-on-Chip (NoC) is an embedded switching network to interconnect IP modules that acts as a promising scalable solution for such a communication bottleneck.

The design space of NoCs is considerably larger when compared to a bus-based solution, due to different routing and arbitration strategies as well as different topologies of the communication infrastructure. Inherent redundancy establishes fault tolerance and helps in dealing with the communication bottlenecks. Networking theory and methods that are applied by NoCs to on-chip communication, bring notable improvements over conventional bus and crossbar interconnections. This improvement enables the SoC designer to find suitable solutions for different system characteristics and constraints.

A network-on-chip is composed of three main building blocks. The first building block is the network interface (NI), which acts as an interface between the NoC and the resources. It injects outgoing messages from the tile to the interconnect and delivers incoming messages from the interconnect to the tile. The reuse of IP cores in a plug-and-play manner can be achieved by using a network interface, since each IP may have a distinct interface protocol with respect to the network. The type of switching is packet switching and in a message-based communication system, by switching the packets, there is no need for circuit switching. This means instead of assigning a static configuration to each router, we use a packet-based router configuration. In other words, to improve the efficiency of the resource allocation, we divide a message into packets for the allocation of control state and into flow control digits (flits) for the allocation of channel bandwidth and buffer capacity.

Routers are another building block of NoCs and realize a communication protocol between processing elements. The main contribution of the router is receiving and forwarding of packets. The packets are received from the NI or the adjacent router and are forwarded to the attached NI or another router according to the routing algorithm and the destination that is
defined in the header of each packet. The topology of the NoC can be defined according to the number of input and output units at each router and the connection pattern of these units (e.g., mesh, torus, folded torus, hypercube, octagon). During the transmission of a packet, different situations are handled by the router, deploying a set of policies. For instance, the deadlock and livelock avoidance, the reduction of the communication latency or the increase of the throughput.

The last building block is the set of links that realize the interconnection among the tiles and act as a glue element among NIs and routers.

The SAFEPOWER architecture supports different types of routing in a NoC, each one leading to different trade-offs between performance and cost.

- **Source-based routing:** Source-based routing can be used as an optional routing method to establish determinism for the whole messages to prevent the event-triggered messages to interfere with time triggered messages. In source-based routing, the sender of the message determines the path through which the message traverses routers to reach the destination.

- **Distributed Routing:** In distributed routing algorithm, calculation of routing function will be done in each router. The header stores just the destination information of traversing packets across the network. This type of routing is favourable for regular topologies because all the routers use the same routing algorithm. XY routing is a kind of distributed deterministic routing algorithms which is the most popular one. The address of each node can be defined by its coordinates (x, y). This algorithm in each router compares the destination address with the current address which is stored in each header file. First the current X address and X Destination are compared and then the West or East port will be selected. If these two X addresses are equal then the Current Y and Destination Y will be compared and according to the result South or North port will be selected. If these two Y addresses are the same, we are in the correct destination and the packet can be routed to the core port of router.

### 1.3. Schedule

Both the application and the physical model are brought together to intertwine by a schedule, which provides a temporal and spatial allocation of hardware resources to applications.

Integrating applications on a common hardware platform using shared resources requires a deterministic behaviour as mentioned in previous sections. The system must provide guarantees, that it always performs in a safe way such that no lives might be endangered. A basis therefore is the progression of time controlling task execution and communication. The timing of these activities is usually defined at design-time in a schedule deciding at which instant a task is executed or a message is sent.

The precompiled resource allocation defined in a schedule is a convenient tool to ensure temporal and spatial partitioning. A schedule is a solution of the *NP hard* problem of arranging multiple tasks to all perform within a given period of time while meeting their associated restrictions, e.g. deadlines or interdependencies. It assumes the WCET when planning the task
execution, therefore granting all applications enough time to finish the task in the assigned timeslot. In mixed criticality systems WCET assumptions can vary based on the criticality level. Low criticality tasks can be assigned an optimistic time slot, as faults can be acceptable in these tasks. High criticality tasks will always be assigned very pessimistic time frames, as the tasks have to be performed at all costs as reassigning allocations too early will cause fatal system failures.

By scheduling all applications a priori at compile time all applications can be guaranteed to be executed within their given deadline. Furthermore, the schedule also plans when and by whom shared resources will be accesses to avoid conflicts or that a low criticality application could access resources from higher criticality applications. Our SAFEPOWER system also provides an on-chip communication schedule where each message injection is preplanned and therefore not only message collisions are preempted but the system can also guarantee a timely upper bound for each message transfer which is important for the system verification.

This interdependence between the execution schedule and the communication schedule lead into a co-scheduling problem. The communication must provide message slots for the tile in reasonable timeframes after the tile’s applications are sending a message. At the same time the hypervisor can schedule its partitions to perform the application according to the communication schedule. In predefined static systems, this interdependence is not troublesome, as the problem gets solved at design time and does not affect the system at runtime.

1.3.1. Schedule Domains

To perform all tasks and transmissions within the system according to the preplanned schedule the execution and communication has to be time-triggered. Using a time-triggered execution of events is only depending on time, whereas unexpected events or interrupt cannot be used to trigger computations. On the execution service level the schedule defines when a partition is mapped to the core and when it is put on hold. The partition internal schedule is then performed by the application itself or the corresponding operating system (OS). On the communication level, the schedule defines when each message will be injected into the NoC. These two schedules are detached from each other but depend on each other, the areas they govern form two scheduling domains divided by the network interfaces. The execution schedule domain covers the tile and the communication schedule domain the NoC. The network interface poses as a time contract where the “meeting times” of the two schedules are defined, because if the execution service wants to send a message a tick later than the communication schedule expects the message, it will be delayed until the next scheduled message injection. This can be fatal for real-time applications, to avoid problems the two schedules will be computed together as a co-scheduling problem. We introduce the term time contracts for the interfaces defined within the network interfaces declaring where the two schedule domains meet.

1.4. Meta-Scheduler

The approach of SAFEPOWER is to add dynamic behaviour by computing several valid schedules which will be chosen based on the system status. For this, a meta-schedule has to
be defined which defines the changes between various other schedules. This meta-schedule must describe which conditions to be met to trigger the current schedule $S_i$ to change into a more appropriate schedule $S_j$. Each schedule $S_i$ is a finite set of components $K_i$ and describes a complete application behaviour as defined in Equation 1. The decision if a schedule should be changed will be based on the context, a finite set $C$ containing variables describing the system status, e.g. battery status $b$, task performance $p$, and temperature $t$. This context information is time-sensitive and therefore any reaction to it must be done immediately, especially the slack information loses its value with passing time. It is apparent that the context must show the global system state, since the schedules are computed for the whole system and global task interdependencies must be taken into account. The system will therefore follow a state machine in which each node describes a valid schedule and each edge represents a context change. Each node will have the same application models, with the same set of tasks but slightly different timing values for the tasks’ start time and execution time. The task deadline will always stay the same, as the criticality requirements must be met in all schedules.

Based on Health monitors, the schedules will be changed and each schedule provided a safe execution plan for the tasks. Each schedule is computed for a certain context and is optimized to minimize the energy consumption for the given environmental context. The optimization can not only lead to lower frequencies and corresponding slower execution times, but the schedule’s context can also lead to non-critical tasks to be dropped. Dropping minor tasks can help the system to prolong the lifetime for the critical tasks and therefore the safety of the system.

The input for the meta-scheduler must be defined by the user at design time. The following input models are used in the meta scheduler (see Figure 2):

**Application model:**

The application model defines all tasks that need to be executed at runtime. Each task must be defined by its starting condition, e.g. dependencies to other tasks or a specific time and the tasks worst case execution time (WCET). Additionally, to ensure that the scheduler realizes the criticality level of the task, each task will be assigned an execution requirement. Low criticality tasks will be for example associated with a certain minimal level of energy that needs to be supplied to the system. This feature is depending on the systems design and which resource is considered to be the most valuable one. If a system has a constant energy supply the low criticality tasks may still be shut down in favour for the system lifetime, the reason to do that need to be provided to the system.

**The Platform Model:**

The platform model contains all hardware tiles and links. Each tile contains a definition of reachable features like the available voltages and frequencies the tiles can run at. Moreover the logical placement of the tiles in relation to each other is defined as well enabling the scheduler to search for optimized communication paths. The links are defined as a weighted graph with the weights being used as additional input to the scheduler’s optimization problem.
The Context Model:

The context model defines the monitored system and environmental behaviour. In the context set, each monitored value that can cause a change in the system must be defined. Monitored values can be e.g. the system’s temperature or the battery level. The context can also include software monitors like slack or fault detectors.

![Image of the Context Model]

Figure 2: Meta Scheduler Model: the Application, Physical and Context Models are used as input for the meta-scheduler, who iterative invoces a scheduler to compute a set of Schedules which form the adaptive System Model

As mentioned before, the determinism of a decision on changing a schedule and its timing is vital for the system’s safety assumptions. All tiles must unanimously decide to change for the same schedule, or decide not to change a schedule at all. To ensure this behavior the meta-schedule will provide global trigger-contexts that, if the tiles detect those, will make them choose the corresponding new schedule. The meta-schedule implements a state machine that has defined target states for all possible context inputs. All tiles have the same global state machine implemented, though for memory purposes their schedule information is reduced to the local information actually needed to perform the adaptation. This means that each tile will have a local extract of the global state machine, which is triggered by the global triggers.

Since all tiles will decide a schedule change based on the same input, the agreed global context information, all local decisions will end in the same new state. By putting these strict restrictions on the underlying schedule the system-wide determinism will be guaranteed.

1.5. Adaptive Architecture

Like in most areas, the choice of an optimal energy management is highly dependent on the application the system has to support. There are applications in which the tasks are performed predominantly independent from each other with each task being assigned to a dedicated processor on a specific tile. While the hardware is still interconnected by a bus or network on chip, the tasks are not interdependent on each other. These systems do not need system wide applicable low power methods, but services which enable each processor to manage itself in the most energy efficient way and in this way, enabling the hypervisor to switch their respective partitions’ scheduling at a high rate, resembling dynamic adaptation to the systems context. This adaptation would only be performed within the systems tiles, with no impact on the rest of the system. This local adaptation doesn’t affect other tiles in a way that their
execution could be stalled by abiding to the globally set timing contracts for resource access and communication. This way the local tile performance can be optimized without impacting the performance on other tiles to guarantee their real-time requirements.

Other systems have highly interdependent tasks where a global adaptation scheme which would rewrite the time contracts can save far more energy than just local adaptation. A detailed motivation for the two system types will be given in section 1.5.1. These systems need methods to adapt to the system state and apply the globally optimized power saving functionalities. To provide solutions applicable to both scenarios, the SAFEPOWER architecture offers two kinds of low power services which cover the local scope, given by the tiles, and the global scope in which the low power features are applied on the whole system, including the tile and the NoC (Figure 3).

Using these two architecture flavours, SAFEPOWER is able to provide the best low power features to all system types. Local Scope Systems which focus on the tile can use a subset of the available low power methods which are specifically optimized for the local use on a hypervisor managed tile. While global scope systems will be provided with services which allow the system to adapt towards a global energy saving maximum by applying energy methods through the whole scope.

1.5.1. Motivation for the two Scopes

At present, the prevalent techniques for energy management are focused on the local tile applying mature methods, e.g. DVFS. We want to expand this local energy management to global energy management scheme. By using the local slack in a global optimization, the network can be optimized to a global rather than just the local optimum. This leads to a maximized system lifetime, especially in systems with cross tile task interdependencies. In these systems, the output of one tile is needed by other tiles to start their computation. For
example, a five-tile architecture has dependencies where the first tile computes inputs for the other four tiles which themselves compute inputs for the first tile, as shown in Figure 4.

![Figure 4: Example for task dependencies: four tasks on four tiles are scheduled to wait for a task on tile 1 to finish](image)

If the first task finishes its execution earlier than the assumed WCET, this slack time can be used for energy optimization. A currently often used technique is to gradually check the execution time. When the monitoring software recognizes faster execution than expected, it dynamically reduces the voltage and frequency of the execution. This way, the WCET timeframe is used more efficiently. In our example, local optimization would gradually adapt the frequency on core one. This would extend the core’s execution time under lower frequency to meet WCET as shown in Figure 5.

![Figure 5: Local energy management: When the execution is faster than the schedules WCET, the tile can locally perform DVFS and lower the execution frequency to finish at the expected WCET and avoid idle times in the schedule time block](image)

We cannot just use the slack by sending the message directly to the other tiles because the message injection is bound to the communication schedule. Therefore, in such a local adaptation the goal is to optimize the time slot given by the schedule. This would save energy, but the other tiles do not profit from this approach. In systems with interdependent applications, the later tasks can also profit from slack. Our approach is to use the slack where the energy savings are best: in this case, the slack would be passed to the other four tiles, as Figure 6 shows.
These four tasks could be started right after the first one finishes, therefore giving them more time to execute. The local tasks then can optimize their newly assigned timeslots using the local DVFS. This way we can multiply the saved energy by four. Considering that especially in mixed criticality systems where the highly critical tasks are planned with a most pessimistic WCET assumption, these slack times unnecessarily block resources for other applications. Reconfiguring these access times on-line is another advantage of our approach.

The reconfiguration on the local scope will focus on the local schedule domain, meaning that all adaptation must ensure to meet the NI time interfaces. This restriction decreases the potential energy savings of the local scope. Still it is easier to implement due to the maturity of the low power features within the processor and could be applied in COTS industrial application with ease compared to the yet unprecedented global adaptation. Within the SAFEPOWER project we will determine how this adaptation can be optimized regarding the context definition and synchronization as well as the applicability to different types of topologies.
1.6. Execution Services

The platform is used to support a set of mixed criticality real time applications. These activities are supported by hypervisor partitions, that from one side provide spatial and temporal isolation, and on the other hand offer a set of services that allows the engineering of applications over the foresaw platform. Depending on the character of the service and the hardware platform, it is provided by the hypervisor (XtratuM) or by the Abstraction Layer included in the partition (e.g. DRAL (DREAMS Abstraction Layer)).

1.6.1. XtratuM hypervisor

XtratuM is a real-time hypervisor that provides, by means of para-virtualization and full-virtualization techniques, multiple isolated execution environments called virtual machines or partitions. XtratuM is intended to be executed in a bare metal computer.

XtratuM provides a hardware abstraction to allow the partitions to execute its code as in a native bare machine. Depending on the support provided by the underlying hardware, part of the hardware could be virtualized, para-virtualized or emulated.

The following are functions, properties and services provided by XtratuM:

**Spatial isolation:** A partition is allocated in an independent and unique address space. The hypervisor guarantees that any other partitions cannot access to this address space. This restriction could be relaxed by the system integrator, if shared memory regions are defined between partitions. The spatial isolation requires a hardware support to be achieved.

**Temporal isolation:** Each partition should be executed without external interferences of other partitions. XtratuM provides a Cyclic Scheduler that executes a predefined plan composed by a set of temporal slots, each of them assigned to a unique partition. The hypervisor guarantees that a given partition is executed inside their assigned slots independently and without disturbances. The hypervisor configuration defines a set of Cyclic Scheduler Plans, and a service is provided to switch between them.

**Fault tolerance:** XtratuM detects, reacts and logs the anomalous events or states of the system by means of the Health Monitor (HM). This is a module of XtratuM that could be configured to catch application unexpected behaviors and malfunctions. Once the HM detects the event, it reacts applying a preconfigured action and, if required, logs the issue.

**Time services:** XtratuM manages directly some devices like timers and clocks. Therefore, the hypervisor offers them as virtual devices by means of para-virtualized services. Concretely, XtratuM provides:

- 1 Hardware clock that represents the system’s running time.
- 1 Execution clock that represents the partition execution time.
Timer that could be set to generate interrupts.

**Interrupt and Exception Management:** XtratuM provides to partitions exception and interrupt management that resembles the one provided by the real hardware. XtratuM virtualizes hardware interrupts that have been assigned to a specific partition in the configuration. Additionally, XtratuM extends the number of interruptions by adding some interrupts related with virtualization services (e.g. Slot Start interrupt).

**Communication:** XtratuM provides inter-partition communications. The hypervisor implements a message passing model between two or more partitions. This mechanism highly resembles the defined in the ARINC-653 standard. XtratuM defines communication channels as logical paths between one source and one or more destinations. Channels are accessible from access points called ports. The hypervisor is responsible to encapsulate and transporting messages. At partition level, messages are atomic; the message is completely read or nothing is received. The hypervisor provides an extended interrupt to inform partitions about events related with their port. Additionally, XtratuM provides communications using communication busses. XtratuM provides a Port interface to the devices that controls the access to these busses. These devices and ports shall be defined in (the XtratuM Configuration File or XM_CF). These devices are accessed only under partition request (when a partition accesses the corresponding port) ensuring the spatial isolation. XtratuM provides two transfer modes:

- **Sampling port:** This transfer mode allows unicast, multicast or broadcast. No queuing is supported. The transmitted message remains in the channel until is overwritten by a new transmit operation. Partitions access the last transmitted message.
- **Queuing port:** This transfer mode support buffered unicast communications. Each port has an associated queue where messages are buffered until they are delivered to the destination partition. Messages are delivered in FIFO order.

**System and Partition State:** XtratuM provides services to partition and system management and monitoring. XtratuM defines two kinds of partitions (regular and systems). Regular partitions can manage and monitor its state and logs. Additionally, system partitions are allowed to perform these actions over other partitions and the systems including switching between system scheduling plans.

**Traces:** The hypervisor provides a mechanism to store and retrieve the traces generated by partitions and XtratuM itself. Traces can be used for debugging, during the development phase of the application, but also to log relevant events or states during the production phase.

### 1.6.2. DRAL

The DReams Abstraction Layer (DRAL) is an interface that wraps the applications with the underlying systems either for an application with or without guest OS, and on top of a hypervisor (e.g. XtratuM partitions) or directly on bare metal (e.g. MicroBlaze runtime). This interface was developed during DREAMS project (FP7-ICT-2013.3.4-610640) on top of the XtratuM. Most of the DRAL services take advantage of the services offered by the hypervisor, but the API has been adapted to be closer to partition needs. Some services could impact the global system, breaking the partitioning isolation, for this reason these services are restricted.
to system partitions, while others do not have access restrictions. Next listing shows the

groups of offered services:

- System Management (restricted).
- Partitioning (restricted).
- Time management.
- Interrupt Management
- Schedule management (restricted).
- Communications Services.
- Health Monitor (restricted).
- IO Management (restricted).

Additionally, in case that DRAL was used in a system without guest OS, it provides the partition
boot and the vCPU initialization.

The API of this abstraction layer can be extended to provide custom features. In this project,
DRAL will be extended with services related to low-power characteristics and it will be called
xDRAL (Extended DREAMS Abstraction Layer).

1.7. Local Low Power Optimization

The execution services are divided into four main groups: power management in an off-line
way based on generation of operational modes, peripheral power management, dynamic
frequency scaling and idle mode management. On the following sections those groups of
execution services are described. The impact of the services on the existing SAFEPOWER tools
is also considered. Is worthy to mention, that the instantiation of the architecture to the Xilinx
ZYNQ platform also consider mapping to the Processing System – PS (ARM side of the
platform) and Programmable Logic – PL (FPGA side of the platform).

1.7.1. Power management based on operational modes

In a mixed criticality system, the applications have different requirements in the operation
and life of the system and the completion of some tasks must be guaranteed even at the
expense of others. Typically, the specification of operation modes is only based on functional
characteristics and external stimuli, but the energy budget can also play an important role in
the selection of tasks and creation of scheduling plans.

In this way, the energy depiction for the final system could be a new requirement in the
development of mixed criticality applications, where in addition to the analysis of the intrinsic
hardware consumption, the operational execution modes must be characterized by a power
consumption profile as it is shown in Figure 7.
Special plans could be defined to address critical energy situations and in the same way, the plan could determine what peripherals are allowed to remain in use.

**Impact:**
- System specification: hypervisor configuration file

### 1.7.2. Power management for peripherals

The devices that have available clock gating mode can expend this feature on runtime. However, the usage of clock gating depends on the allocation of the devices to the hypervisor or the partition.

The hypervisor could manage directly the power consumption of the peripherals reserved by it. However, the devices allocated to the virtual machines will be the responsibility of the partition application (guest OSs or execution environment) in order to put the device in the appropriated power mode in the duration of the slot.

At boot time, the hypervisor in combination with the bootloader would guarantee that the devices unused by the hypervisor would be powered-down until the respective partition activates the device allocated. When partitions with devices allocated are not in execution, i.e. when the partition is not active due to scheduling decisions, the I/O server or the owner partition of the device can decide to power-down those peripherals before suspending the execution. Previous state of the peripheral would be restored when the partition is again active.

The hypervisor can implement low-power modes that can deactivate groups of devices allocated to partitions, but this deactivation is not related to an explicit disabling of a device.

I/O server partition is the best way to manage the power consumption of the devices in the hardware platform.

**Impact:**
- I/O server partition: management of power modes of the devices.
1.7.3. Idle management

Taking into account that the hypervisor provides cyclic static scheduling, from the hypervisor perspective, two types of idle times are considered:

- **Idle plan time (Idle-PT):** This idle time is intrinsic to the definition of the scheduling plan. It corresponds to gaps within partitions’ execution previously defined in off-line scheduling analysis.

- **Idle time slot (Idle-TS):** This idle time is the result of early terminations in the execution of the application during a partition time slot. When the scheduling is generated, a partition slot is allocated based on the worst case temporal needs of the partition. Therefore, on several instantiations of the partition, the actual execution time could be less than the time pre-allocated in the plan.

These types of idle modes imply wasted CPU time and energy, therefore those time slots require some management from an energy perspective.

For this management, there exist several options based on the definition of low-power states or consumption levels for the whole system, for example:

- Light and quick wait state of the processor
- Reduction of processor frequency + light and quick wait state of the processor.
- Light suspension of the Processor + dynamic power reduction of the cache (standby mode)
- Processor + caches + clock gating for some peripherals
- Hard system power reduction through powering-down of the PS peripherals, PL, etc. the architecture or the application itself backups the status in volatile and non-volatiles memory.

These low-power states must be characterized temporally, i.e. these states must have defined entry and exit worst case times for each power management mode.

In these idle times, the processor is returned to the hypervisor which in turn decides which low-power state must be activated. This decision is based on the amount of idle time before the next activation and the temporal and functional dependencies among tiles.

**Impact:**

- System specification: Definition of system low-power modes
- XtratuM API: services of partition informing the hypervisor about early terminations in the execution of the partition.
- XtratuM configuration: define in the hypervisor configuration the low-power modes available to be used on idle-times.
1.7.4. Processor Frequency scaling

The processor frequency can be dynamically changed to reduce the energy consumed by the PS.

However, the changes in the processor frequency affect the execution time of the tasks which must be taken into account especially in critical partitions, where unexpected changes in the execution can lead to missed deadlines.

The proposal consists of defining two levels to control the operation frequency:

- At hypervisor level: where static allocation of frequencies is defined in the system specification (Configuration File - CF).
- At partition level: processor frequency is modified locally based on restrictions imposed by the hypervisor.

In the definition of the configuration file each partition specifies a maximum processor frequency (MPF) and a base operational frequency (BOF). Additionally, another partition frequency can also be specified by time slot in the scheduling plan. That slot frequency (SF) must be less than the MPF. The operational frequency is used by default in all activation of the partition that does not define SF.

At partition level, the processor frequency could be changed locally in runtime for the current slot. The hypervisor would provide a service for such action and the hypervisor would check the restrictions imposed in the CF. Frequency changes would affect only the current slot, i.e. frequency changes would be maintained only during the period of the current slot. In the next slot, the frequency is restored to that defined in the CF.

Impact:
- System specification: XtratuM configuration file
- XtratuM API: services of partition
GLOBAL SCOPE

The global scope follows the same adaptation principles as the local scope: a meta-schedule predefines which schedule changes are taken under certain environmental and intrinsic statuses. This section explains how the global adaptation translates the local tile adaptation into a global optimization.

In the global approach the locally established system states are accumulated into a global system view. By establishing this global context knowledge, the system can switch its behaviour into a globally optimized direction. The adaptation is performed in three steps:

First, the global state is established by a protocol. Based on the global context the communication and the execution schedules will be marked for a change at a certain time. When the change-time comes, all tiles and network interfaces change into the new schedule, ensuring a globally consistent state.

1.8. Adaptation architecture

The adaptive services will be performed locally, meaning that no new centralized manager module is needed where the scheduled change decision will be done for all tiles. One could argue to perform changes based on the local information alone. The main problem with a distributed changing scheme would be the unavoidable inconsistency of the system. Local context can not only vary but can contradict each other. The jobs were scheduled at design time to be optimized to a certain goal and taking global task interdependencies into account, changing a local core’s behavior can impact critical tasks on other tiles by delaying a vital message which in turn may lead to missed deadlines. In the safety critical systems which SAFEPOWER aims to be applicable to, these deadline misses can endanger people’s lives and must be avoided at all costs.

Additionally, since the adaptive communication works based on a globally computed schedule where for each trigger, all tiles will make the same scheduling decisions, one can see it is essential for such a setup that all tiles have the same input to decide upon. This means that each local adaptive communication service needs a collection of all other tiles’ local context information. This information needs to be distributed to the other tiles and agreed upon before any changes can be made, thus providing a global knowledge to all tiles.

To ensure this global knowledge the SAFEPOWER architecture will provide a system wide agreement protocol.

To fulfill the service requirements defined in D1.2 “Analysis and selection of low power techniques services and patterns”, the hardware block must be able to support three functionalities:

1. Must be able to retrieve its tiles’ context information.
2. Must be able to communicate to other tiles to send its own information and receive the local status of the other tiles. Therefore reaching an agreement on the global status.
3. Must be able to change the tile’s local schedule and initiate changes in the core and NoC.

The additional services will be embedded in the NI functionalities and located between the core interface and the Bridging units. This way the adaptive services can inject messages to be routed via the NoC to reach the other tiles and at the same time communicate with the core to collect context data.

To perform these tasks the adaptation building block will be located between the core interface and the bridging units. This interface will be used to receive information from the local core. The agreements send the globally agreed context string and a timestamp to the adaptive communication layer. The size of the transmitted bit string with the context information will vary based on the monitored context information. As the global context will represent all tiles’ local context information, the size of the string will be equal to $n*(\text{sum of all context bits})$ with $n$ being the amount of tiles.

Furthermore, the agreement layer can use the core interface to send and receive messages over the NoC. These messages with have the following setup:

<table>
<thead>
<tr>
<th>Dest ID</th>
<th>Src ID</th>
<th>Data</th>
</tr>
</thead>
</table>

Figure 8: Message layout between agreements layers

With Dest ID being the destination ID for the neighboring tile, Src ID will mark the sending tile and the Data bits are used to transmit the context information.

The agreement shares an interface with the adaptive communication to be able to send the agreed state to the adaptive communication, where the decision on a schedule change is made. The adaptive communication will decide on a possible schedule change and inform the bridging units using the interface to the scheduler within the bridging units. If the adaptive communication decides to change the schedule it will transmit a $\Delta$ to the new schedule and the tick in which the schedule change shall be performed by the NI.

1.9. Communication services

We first describe the basic communication services which the SAFEPOWER architecture provides in 1.8. Based on these services the adaptation on the global scope will be described in this section.

1.9.1. SAFEPOWER NoC Services

In this part, different services of SAFEPOWER NoC (SPNoC) based on the concept of safety and power efficiency are introduced.
1.9.1.1. Architectural style

The architectural layers of SPNoC are based on OSI layer as illustrated in figure 6. Each layer offers a set of services and can be mapped to one network layer. Before describing the architecture details, an overview of the network layers is provided.

![Diagram of SAFEPOWER NoC Services compared to an OSI Layer Model]

Application layer in this context stands for the execution services which have been described earlier in Section 5.1. This layer provides the connection to the lower layers. Tiles and Monitors can be mapped to this layer. Slacks are detected at the core level by dedicated HW/SW units, as it will be described in 7.1. The Network Layer is mapped to the network.
interface (NI) and provides communication services which will be illustrated in detail by the next section. This layer is responsible for providing packetization and de-packetization of flits between the tiles and the NoC. This layer also establishes the mapping between the tasks running on the tiles and the communication channels\(^1\) of the NoC. Data Link Layer is made with the purpose of extracting route and transferring packet across physical layer. Routers functionality can be map to this layer. Finally, the lowest layer serves as interconnection for moving data from output of a switch to the corresponding input of next.

### 1.9.2. NI services

The NI makes the core and the communication infrastructure independent of each other. NI can serve as a sender or receiver. A sender NI determines the path and concatenates the head flit, the body flits and the tail flit together and will send it as a packet to network. NI as a receiver NI, generates the messages out of received flits and provides the processor cores with the messages.

As shown in Figure 9, the services provided by the SAFEPOWER NI can be divided into 5 different groups as follows:

1.9.2.1. **Network interface front End**

The Network interface front End is core interface and acts as the interface between the cores within the tile and the NoC through ports. Each port establishes partitioning, thereby supporting mixed-criticality in case of different type of messages. The NI front end also provides the application layer with status and reconfiguration registers.

1.9.2.1.1. **Ports**

Ports can be divided into two types: input and output ports. In case of an output port, the processor core writes the message into the respective outgoing port to be delivered to the NoC and in this level the destination can be implicitly declared with port configuration operation. In case of an input port, once the message reaches a destination NI, it will be placed at the input port to be read by the respective processing cores.

Each port has the semantics of the state and the events ports. State ports will be used for buffering states or recording events in FIFO depending on what we need to store.

Each port also applies starting and ending points of the Virtual Links. Virtual Links (VLs) are an abstraction over the networks and hide the physical system structure of the platform from the components.

1.9.2.2. **Agreement layer**

\(^1\) This concept is known as “virtual link” in the context of the DREAMS [5]
As will be described in section 6.3.2 the Agreement layer propagates local context information to all the tiles of network in order to set the system to a global state and determines agreement time definition.

1.9.2.3. Adaptation Layer

As described above, the contribution of the Adaptation Layer is to ensure a global system state in Network on Chips. The decision making for new schedule by each node will be determined by this layer and this new state will be propagated in whole network in order to sync tiles to change pervious schedule to the new one.

1.9.2.4. NI Core functionality

This Layer proposed based on the DREAMS Project to fulfill requirement of safety and mixed criticality features. Safe Power Network Interface supposed that three type of communication can be supported as follows: periodic injection of time-triggered messages, traffic shaping of rate-constrained messages and Relaying of best-effort messages. In this layer also monitoring service provides the application layer with errors and statics.

1.9.2.5. Network interface back End

This layer will be defined on the underlying network on chip. The main task of this interface is packetization and depacketization of flits. The packetization unit can build a header flit including the path and destination information based on the information provided by the registers and, then, wraps up the head flit and body flit into a packet. On the other hands, depacketization unit can select head flit from the head flit and assert an interrupt signal to the core. This layer is also responsible to provide header encoding and decoding services. Each header carries some information so the mechanism of releasing information from each header or collecting them to header is expected from Network interface back end layer. In the case of source base routing path can be looked up in this layer. DVFS control mechanism can be supported by this layer.

1.9.3. Router Services

The cross connection between network interfaces which built based on the network on-chip in combination with the physical links can be realized with routers. The routers get head flit and then relay the body flits according to the configuration obtained from the head.

Each router is composed of input and output units, switch and control logics, configuration information which collectively implements the flow control functions required to buffer and forward flits to their destinations. We will examine the services a typical on-chip router provides with regards to prioritization.

The services which are provided by the routers can be categorized as follows:

1.9.3.1. Switching
Each flit of a packet needs to traverse through the switching unit and this unit is fully configured by the Switch Allocator (SA). The responsibility of switch allocator is to allocate switch slot per each flit. It means that each flit needs to request a time slot of the switch from the SA.

1.9.3.2. Priority base channels

In order to guarantee bounded delay and low jitter in the network, the proposed router can support priority base channels. In this case, there are three main priority classes in the architecture, each of which can possibly be composed of different further priorities. The highest priority class in the network is set to the periodic messages. For periodic messages there is no priority needed and messages are sent according to the predefined schedule. The second priority class which have different levels of priorities is assigned to sporadic messages so two sporadic messages of different priorities can compete for using a resource. In this case, the one with the higher priority will win and the other will wait. The lowest priority class is assigned to aperiodic messages. There are no guarantees about whether and when these messages arrive at the destination. The number of priorities can be modified according to the implementation.

1.9.3.3. Route extraction

Execution of route extraction is defined in router. In case of source base routing it can be calculate base on header flit of each packet and in case of distributed routing it can be extracted according to routing algorithm in each router. And also SAFEPOWER routers can support Wormhole switching in order to achieve more efficient use of buffers and bandwidth.

1.9.3.4. DVFS Control

The need to improve efficiency of integrated circuit has led us to use Dynamic Voltage-Frequency Scaling (DVFS) techniques. A control mechanism of DVFS is proposed to control and change voltage and frequency transient periods according to the situation in each router.

1.10. Global context establishment services

The optimization of local tiles will always optimize the energy usage to a local optimum, as the adaptation can only react on the status known to the core itself. By relaying this local information to all other tiles the system can optimize its energy usage to a global maximum thus increasing the system lifetime. This chapter focuses on such a global optimization,

1.10.1. Establishment of global context

The architecture’s underlying platform supports various tiles which will be performing their dedicated applications in complete separation from each other. The only connecting element of the system is the network on chip (NoC). Our adaptive services will implement a layer which will gather system information using the network on chip resources and enabling the tiles’ NIs to give this information as feedback to their dedicated tiles. This way the isolated tiles can be managed based on a globalized view on the system.
Having the globalized view, the tiles will be enhanced with adaptive services which will allow the NI to change the schedule it is performing by. Each schedule will be verified at design time and will be optimized to save energy. The schedule changes will be applicable at periodic rates to prevent event-triggered schedule changes, with the rates being optimized to an optimal frequency, since too many schedule changes in a short time can lead to a higher power instead of a lower one.

Using the NoC, information about the other tiles and the system status can be exchanged. By providing all tiles with a global system state each tile can manage itself locally and no additional global management module will be needed, which is favorable from an energy saving point of view. Allowing the tiles to change based on local information only will lead to discrepancies in the applied schedules. If only two tiles decide to switch to two different schedules, neither the timing of the communication can be guaranteed nor the avoidance of collisions.

1.10.1.1. Global knowledge

The adaptive communication services enable the system to change its schedule based on its current global status called context. The context is defined as a finite set of attributes $C_i$ that express the system's status, slack and battery health for example. Some of the context types can be used to improve the system's energy usage by reallocating slack times to later time slots, allowing the tasks to run at lower frequency. Other context types indicate that the system needs to go into a low power mode to prolong the system's lifetime until maintenance or recharging functionalities can improve the battery health again.

These environmental and intrinsic status values can change at any moment during runtime, therefore efficient energy saving mechanism must reconfigure the current system instantly to make use of the occurring context events. Having such unpredictable behavior is impossible for a certifiable system. Therefore, the SAFEPOWER approach is to have a semi-dynamic schedule change, where the whole system follows a state machine which defines the system's schedules and will change into other schedule based on its triggers. The triggers in this case will be the previously described context.

The context variables are bounded, as only the ones considered in the meta-scheduler will be monitored in the running system, since the state machine only regards them for changes and is able to adapt solely to these situations at runtime. The context values will be collected via sensors, which will be provided in hardware as well as in software. The hardware sensors like for example thermometers and the battery monitor will be directly connected to a small subset of agreement layers to avoid that each tile has to poll the information. To be fault tolerant, these monitors will be connected to at least 2 tiles at the same time. Other monitors will be provided by software functionalities of each corresponding core. The core will for example inform the adaptive services when jobs have been finished, to enable them to compute the slack.

1.10.2. Agreement
As mentioned before the agreement period is vital to ensure a global system state. The Agreement protocol will be used to propagate the local context information from all tile’s to all other tiles. In classic networks these kind of information would be send in a broadcast for every tile to receive. As NoCs usually don’t provide broadcasts the propagation of information to all network members is more difficult.

There are two major issues when implementing a system wide agreement:

1. The local information is volatile and loses its value after a short period of time, therefore the agreement must be performed as fast as possible a certainly short time.
2. The moments when a schedule change will be performed must be deterministic for all tiles.

Regarding point 1., the Agreement will have to use a bounded time therefore the message path between the tile must be known beforehand to make timing assumptions. This excessive time frame must be considered in possible schedule changes. If for example a task has a slack at time \( t_s \) the schedule change can only be performed after \( t_1 + t_{agreement} \) with \( t_{agreement} \) being the worst case agreement time (WCAT) which is the time it takes the system to agree on a global context.

The second point calls for a second agreement round where all tiles will agree on a schedule change time. This could be realized by defining a default waiting time that has to pass before a schedule change is induced. In this time each local adaptive communication service has the time to decide if a schedule change is performed. An alternative is a precompiled schedule change time that is defined in the meta-schedule. In this case the tiles would know that if they have a certain trigger, they have to initiate the schedule change at the corresponding changing time. To decide on an implementation, the trade-off between scheduled times and a second agreement run in term of powers loss and power saving must be analyzed.

1.10.2.1 Agreement information propagation

For the propagation of the local context information an atomic broadcast will be formalized that will initiate the agreement periodically at a predefined rate \( t_{p} \). All tiles will send their local information and receive the information of all other tiles. The protocol will further ensure that the network will not be congested by the message load and the agreement will be performed within a timely bound \( t_A \) with \( t_A < t_{p} \). After the agreement time has passed the tiles will individually decide if the schedule needs to be changed. The decision will be deterministic and unanimous for all tiles as it will be based on the meta-schedule provided by a scheduler at design time.

The propagation of the agreement information can be performed via the NoC of the system or it can be done via a second dedicated network. Using the NoC has the advantage that no additional wiring is needed. Considering that the additional links will have to be powered as well, the energy consumption would increase initially compared to an energy-unmanaged system. The trade-off between additional wires and power savings from the schedule reconfiguration will have to be analyzed in later steps.
Using the NoC will on the other hand result in a frequently high network load that would disturb the normal system communication. Looking at the time-triggered message injection these periodic message bursts will need to be scheduled within the normal traffic, complicating the scheduling procedure.

The choice on how to implement the agreement service will have to be done based on simulations which compare which system configuration would lead to the higher benefits.

### 1.11. Decision making services

The decision on the changes in the schedule will all be predefined in the meta-scheduler. The adaptation layer will only perform changing operations according to the state machine which follows the meta-scheduler. This way, no unpredicted state can be reached and the system is always guaranteed to be in a verified state.

After the global context is established, the adaptation layer will invoke schedule changes in the local tile’s hypervisor and the Scheduler of the local NI, as can be seen in Figure 10. For this, the adaptation layer has dedicated interfaces to the monitoring partition of the hypervisor and the local NI’s dispatcher. After receiving the new schedule information, the monitoring fetches the corresponding schedule changes for the tile and the dispatcher injects the time-triggered messages according to the new communication schedule.

For the changes on the execution services, the adaptation layer uses a port to feedback the new schedule information to the monitoring partition. When the possible schedules are precompiled the tile’s possible hypervisor schedules will be stored within the monitoring partition. The partition is in charge of the local scope’s schedule adaptation and will perform the changes also for the global scope. To save memory space the adaptation layer will only indicate to the monitoring partition which new schedule needs to be applied and the monitoring partition will then be in charge of performing the needed changes.

For the adaptation on the communication layer, the adaptation layer directly accesses the scheduler which is located within the NI’s Core functionality Layer. The adaptation layer changes the schedule directly.
1.11.1. Low power communication services

In the case of NoC, a reliable, feasible, and cost-efficient implementation can be achieved by reduction of power consumption on the communication infrastructure. However, if the network and routers are not designed properly, NoCs can dissipate power and degrade throughput. Up to 36% of the total power consumed by the entire system has been consumed by NoC [4]. Low power techniques can be addressed at different levels of abstraction from transistor level to system level. In SPNOC, control flits (generated by the NIs) can configure the routers in terms of the clock speed.

1.11.1.1. DVFS on routers

One of the most significant techniques to address the dissipation of power is DVFS in Routers which change the voltage and frequency of each router in real time. Each router can scale frequency individually from high power to low power.

1.11.1.2. Router Gating

This technique is used to turn off routers in order to decrease power consumption in Networks on Chips. In case a router is not used during the period, the router can be switched off by the control unit and can be turned-on later with arrival of new incoming packets.

1.11.1.3. Different Frequency areas
NoCs can be designed to operate at different clock domains through router categorization. This categorization is based on the predefined schedule and paths to slow down the clock speed at selective routers.

1.11.1.4. **Adaptive routing based on temperature to avoid hot islands**

Avoidance of hot islands in NoCs has a significant effect on power efficiency. Awareness of temperate in each router and use of adaptive routing can help to avoid heat islands.
MONITORING SERVICES

The basic idea of the monitoring framework is to monitor the current system status to enable behavior adaptation according to the evaluated system status. In addition, a health monitoring and safety management partition (HM) detects and reacts to abnormalities in the behavior of the system.

As seen in Chp.0, the basic SAFEPOWER architecture contains tiles with local adaptive communication services and local health monitors. Each local monitor will be assigned to a core and reports its status in a dedicated monitoring partition.

1.12. Monitors and Monitoring services

Health Monitors on each core provide status information and are capable in case of errors to drive the system into a safe degraded mode.

Figure 11 shows the typical flow in XtratU hypervisor (refer to Sect.1.6.1) in handling health monitoring events where depending on their nature (fault, error, failure), HM actions are undertaken to drive the system into a safe state. This scheme will be adapted for handling HM in the SAFEPOWER reference platform and will be extended with power-relevant monitors. Examples for HM events would be internal sanity checks of partitions, hypervisor state checks and processor exceptions. As seen in the figure, the monitoring partition is able to read/seek/open an up-to-date HM log stream of a specific HM via provided API.

In order to enable the system to manage energy and power, the above HM must be extended with power-relevant monitors. Table 1 summarizes the properties of the extended power-relevant monitors used in the SAFEPOWER reference platform. These are basically of three types:
Shack monitor: detecting slackness of computation slots. With the help of this monitor deadline misses can be detected. In addition, this slackness size is also relevant for the decision of the appropriate low-power technique for e.g. whether to run-to-idle to apply voltage and frequency scaling.

Temperature monitor: monitoring the local and the ambient temperature of the SoC relevant for detecting critical temperature and for low-power analysis.

Power Monitor: monitoring the power rail supply of parts of the platform and detecting critical states.

<table>
<thead>
<tr>
<th>Monitor</th>
<th>Locality</th>
<th>Hardware</th>
<th>Property</th>
<th>Example property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slack monitor</td>
<td>Tilewise</td>
<td>(shared) Timer</td>
<td>Deadline violation, Global power management decisions</td>
<td>$t_{\text{compEnd}} - t_{\text{compSlotEnd}}$</td>
</tr>
<tr>
<td>Temperature monitor</td>
<td>SoC global/ FPGA</td>
<td>On-chip temperature sensor(s): temperature dependent resistor or ring oscillator for FPGA, ambient temperature sensor</td>
<td>Extreme Temperature detection, Ambient temperature change</td>
<td>Always($\text{temp}<em>{\text{SoC}} \leq \text{temp}</em>{\text{junctionSoC}}$), \nDelta($\text{temp}_{\text{ambient}}$) $\leq$ 0.5 °C/min (over a range of 20 °C)</td>
</tr>
<tr>
<td>Power Monitor</td>
<td>Power rails (domains)</td>
<td>Voltage and current Sensors</td>
<td>Valid operating mode, Power limit, Energy budget</td>
<td>Always ($\text{bound}<em>{\text{lower}} \leq U</em>{\text{railx}} \leq \text{bound}<em>{\text{upper}}$), \nAlways ($\text{bound}</em>{\text{lower}} \leq I_{\text{railx}} \leq \text{bound}<em>{\text{upper}}$), \nAlways ($P</em>{\text{railx}} \leq P_{\text{limit}}$), \nAlways ($E_{\text{accumulate}} \leq E_{\text{limit}}$)</td>
</tr>
</tbody>
</table>

1.13. Monitors services - Interfaces to network

The gathered information by the health monitors can be filtered and aggregated in the monitor partition (e.g. by combining several measurements, discretization based on specific value ranges). Then in the corresponding monitoring partition, relevant information of the respective core can be periodically provided to the global system scheduler.

SAFEPOWER will provide services for the monitoring of the state of the system. The list of services is called xDRAL (see Sect. ¡Error! No se encuentra el origen de la referencia.) and it
provides software-support for utilising the platform services from the application software. Since SAFEPOWER deals with a platform that host both hardcore processors (e.g. ARM) and softcore processors (MicroBlaze on FPGAs) two versions of the monitoring services will be implemented: xDRAL, for hardcore processors, and xDRALite, for softcore processors.

Figure 12 describes a limited set of the SAFEPOWER monitoring services, those related to power, slackness and temperature monitoring. It is important to remark that high level monitoring services do not depend on the target processor. Thus, from the user stand point, the xDRAL and xDRALite interfaces are exactly the same, even though the processor architecture influences the implementation of the drivers at low level.

![Figure 12: Software structure of SAFEPOWER monitoring services](image)

1.14. Monitoring services on the Zynq SoC

For monitoring the performance of the Zynq SoC, on-chip performance counters are supported, enabling tracing the activity of single components of the Zynq which can also be used to estimate the power consumption of the system. In addition, special on-chip sensors are available and can be used to dynamically monitor the temperature and power consumption of the SoC.
Figure 14 shows a schematic structure of the hardware required to implement the services on a Zynq platform. Even though SAFEPOWER aims at defining a general architecture the implementation of the services at driver’s level is highly platform dependent. Figure 13 shows the refined version of the generic monitoring services (see Figure 12) when adapted for the Zynq MPSoC that constitutes the xDRAL monitoring services. As it can be seen, the software is platform independent except for the drivers’ level.

The available monitoring devices can be divided into four categories:

- **Ring Oscillator**: it is a particular Register-Transfer Level (RTL) circuit that can be implemented inside an FPGA to monitor the temperature at a specified location. A specific ring configuration of FPGA logic gates creates a resonance at a particular frequency and this frequency is sensitive to the temperature of the chip [12]. Reading the frequency of this circuit the ZYNQ implementation of the SAFEPOWER architecture will have access to temperatures of specific regions of the chip.

- **ADC**: Analogue-to-Digital Converter that can be used to acquire information on physical variables (temperature, voltage, current, etc.). For instance, the XADC present inside each Zynq device.

- **I2C devices**: many voltage regulators and temperature sensors can be accessed by an I2C or PMBus channel.

- **Timer devices**: setting up timer to measure slackness

![Figure 13: Software structure of SAFEPOWER monitoring services on Zynq](image-url)
In the following we will elaborate on the monitoring capabilities of the Zynq platform.

1.14.1. Monitoring – Performance Counters

Several performance counters are supported on the Zynq SoC, to monitor the system components at runtime [6]:

- **SCU Global Timer** (PS). The SCU global timer can be used to timestamp system events in a single clock domain.
- **ARM Performance Monitoring Units** (PS). Each ARM core has a performance monitoring unit (PMU) that is used to count micro-architectural events. These counters can be accessed directly by software, through operating system utilities, or with chip debuggers such as Linux Perf or ARM Streamline.
- **L2 Cache Event Counters** (PS). The L2 cache has event counters that can be accessed to measure cache performance.
- **GigE Controller** (PS). The gigabit Ethernet controller has statistical counters to track bytes received and transmitted on its interface.
- **AXI Performance Monitor** (PL). This core can be added in PL to monitor AXI performance metrics such as throughput and latency. Trace functions enable time-stamped AXI traces, such as time-stamped start and end of AXI transactions to observe per-transaction latency.
- **AXI Timer** (PL). This core can be added in PL to provide a free-running timer in PL. This timer is useful for time-stamping events in PL clock domains.
- **AXI Traffic Generator** (PL). This core can generate a variety of traffic patterns to the PS interfaces. When used with an AXI performance monitor, the traffic generator can...
help provide early system-level performance estimates. The core can be used to estimate data-movement costs and validate design partition choices. These performance counters can be used to construct an analytical power/temperature models to achieve a rough estimation of the power consumption of the overall SoC.


1.14.2.1. The Xilinx Analog-to-Digital Converter

The Xilinx analog-to-digital converter (XADC) is an ADC embedded into all Zynq devices that can be used for monitoring applications on the Zynq SoC (with a sampling rate of 1 MSPS Million samples per second [10]). When instantiated in the PL, the PS-XADC can establish connectivity to the XADC via an XADC AXI4-lite interface. The input XADC has 16 input multiplexed channels and 5 of them are dedicated to measuring internal voltages and temperature. Thus VCCINT, VCCAUX, VCCBRAM, VCCDDR and the device temperature can be always monitored in any Zynq device. Since the XADC has a power supply and temperatures sensors, control information of the PS can be monitored. Each of these sensors can be configured to hold minimum/maximal thresholds which when violated during runtime, alarm signals can be issued [8]. The XADC can be configured via an industrial Linux driver or through a bare-metal driver (provided by Xilinx [11]) to ease the monitoring process for the end user [10].

1.14.2.2. Power Controllers and Sequencers

In addition, there exists several mature work to measure the power consumption of some Zynq-7000 SoC instances via external devices. Indeed, power controllers and sequencers can be used to control the voltage levels and the power on sequence of different power rails. These devices are key components of any power control scheme, since they allow both to monitor the state (voltage, current, temperature) of voltage regulators and to undertake regulating actions. They are intelligent devices that host a microcontroller that runs the control and sequencing algorithms. Internal parameters can be accessed and modified by a serial bus connection (usually I2C).

One example of design employing these devices is the ZC702 evaluation board, that mounts 3 UCD9248 digital power controllers by Texas Instruments (TI) to monitor 5 switching voltage regulators, each of them with two channels, for a total of 10 power rails (sample rate of 5kHz). The information provided by these devices can be accessed in two ways:

1. By using TI Fusion Digital Power Studio, a PC based Graphical User Interface that offers an intuitive interface to the device. This tool allows the design engineer to configure the system operating parameters for the application, store the configuration to on-chip non-volatile memory and observe both frequency domain and time domain simulations for each of the power stage outputs.

2. By employing the PMBus interface of the device, connected to the I2C port of the Zynq SoC through a TI PCA9548 1 to 8 channels I2C bus switch.

The first option can be used to configure the power rails at production stage or for laboratory tests. The second solution is tailored for online measuring and controlling.
For detailed setup description with a demo application refer to [9].

A similar solution can be found in the ZC706 Xilinx evaluation board, that mounts a Zynq SoC too. In this case, the board hosts a UCD90120A power supply sequencer and monitor, and the LMZ31500 and LMZ31700 family voltage regulators. Figure 2.2 shows a schematic of the power system. Table 2.2 contains the lists of the components of the power system.

![Figure 15: Schematic of the power system of the Xilinx ZC706 evaluation board](image)
Table 2: Components of the power system of the Xilinx ZC706 evaluation board

<table>
<thead>
<tr>
<th>Device</th>
<th>Voltage</th>
<th>Max. Current</th>
<th>Net Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMZ31520</td>
<td>1.0V</td>
<td>16A</td>
<td>VCCINT</td>
</tr>
<tr>
<td>LMZ31710</td>
<td>1.8V</td>
<td>10A</td>
<td>VCCAUX, VCC1V8</td>
</tr>
<tr>
<td>LMZ31506</td>
<td>1.5V</td>
<td>6A</td>
<td>VCC1V5, PL</td>
</tr>
<tr>
<td>LMZ31506</td>
<td>2.5V</td>
<td>6A</td>
<td>VADJ_FPGA, VADJ</td>
</tr>
<tr>
<td>LMZ31710</td>
<td>3.3V</td>
<td>10A</td>
<td>VCC3V3_FPGA, VCC3V3</td>
</tr>
<tr>
<td>UCD90120A</td>
<td>N.A.</td>
<td>N.A.</td>
<td>ALL</td>
</tr>
</tbody>
</table>

The LMZ31520 power module is a step-down DC-DC solution capable of driving up to 20A load. The LMZ31520 module can accept an input voltage rail between 3V and 14.5V and deliver an adjustable and highly accurate output voltage as low as 0.6V. The LMZ31506 power module is a step-down DC-DC solution capable of driving up to 6A load. The LMZ31506 module can accept an input voltage rail between 3V and 14.5V and deliver an adjustable and highly accurate output voltage as low as 0.6V. The LMZ31710 power module is a step-down DC-DC solution capable of driving up to 10A load. The LMZ31710 module can accept an input voltage rail between 4.5V and 17V and deliver an adjustable and highly accurate output voltage as low as 0.6V. These modules only require two external resistors plus external capacitors to provide a complete power solution. These modules offer the following protection features: thermal shutdown, programmable input under-voltage lockout, output over-voltage protection, short-circuits protection, output current limit, and each allows startup into a pre-biased output.

Texas Instruments is not the only available provider. Amongst the possible manufacturers it is worth to mention Linear Technology, Analog Devices, Maxim Instruments, Microchip Technology.

Finally, if one is only interested in measuring power and temperature without the requirement of a direct control action on the voltage regulators, power monitors are a good solution. Indeed, on-board power monitors usually can provide information on more than a power rail at a time; they do not only measure current and voltage but also the temperature of the power stage or others devices (some components have pairs of pins connected to an internal transistor that can be used to sense temperature). Example components are TMP513 by Texas Instruments and LTC2991 by Linear Technology.

1.14.2.3. Temperature Monitoring

Concerning measuring the temperature on a Printed Circuit Board, many devices use an external PNP transistor to sense temperature, like for instance a 2N3906. Accurate temperature sensing depends on proper PNP selection, layout, and device configuration. The simplest way to measure temperature is to force a known current into the transistor, measure voltage, and calculate temperature. Some components, such as microprocessors, have pairs of pins connected to an internal transistor that can be used to sense temperature.

As already discussed, heat monitoring of power stages can be performed by power monitors and controllers.
Finally, concerning the measure of the temperature inside the Zynq SoC there are three alternative solutions:

1. Use the XADC as already mentioned in this section.
2. Use an external device to monitor the voltage at the internal diode connected to pins DXP (anode) and DXN (cathode).
3. Use a solution based on ring oscillators or on the physics of the metastable state of flip-flops.

The last solution, although more complicated from a design point of view, is the most innovative and allows the temperature sensor to be placed almost anywhere in the PL of the Zynq SoC.

A ring-oscillator basically consists of a feedback loop that includes an odd number of inverters. The oscillation period is twice the sum of the delays of all elements that compose the loop. The frequency of the ring depends on the temperature and it is measured by incrementing a counter. There exists both synchronous and asynchronous implementation: in the first case the counter is driven by the system clock, whereas in the second one the output of the ring oscillator is used as a clock signal for the counter itself.

The idea behind the temperature sensor based on flip-flop metastability consists in forcing a flip-flop in metastable state by injecting a clock on its input which is asynchronous with the main clock. This way, by adjusting the clock, the occurrences of a metastable state increase. As the interval in which a flip-flop remains in a metastable state before going back to a stable one depends on the temperature, the number of metastable states during a fixed time depends on the temperature too. Thus, it is possible to infer the temperature by counting the occurrences of metastable states.
LITERATURE

3. Dynamic Voltage and Frequency Scaling: The Laws of Diminishing Returns
5. DREAMS architectural style
12. Charles-Alexis Lefebvre, Leire Rubio, Jose Luis Montero, "Digital thermal sensor based on ring-oscillators in Zynq SoC technology" THERMINIC 2016