# Fault-Tolerant and Energy-Efficient Communication in Mixed-Criticality Networks-on-Chips

Adele Maleki, Hamidreza Ahmadian and Roman Obermaisser
Chair for Embedded Systems
University of Siegen

***Abstract*—We observe a tremendous trend towards mixed-criticality systems, where subsystems of different safety assurance level coexist and interact. In addition, embedded systems are demanded to be efficient in terms of energy consumption to achieve longer operation time with the same battery capacity. This paper introduces a novel architecture for an adaptive time-triggered communication at the chip-level, which addresses the above challenges. In the proposed architecture, time-triggered communication offers safety by establishing temporal and spatial segregation of the communication channels. In addition, adaptivity enables the communication backbone to adapt the injection time of message according to the real execution time of computational tasks, thereby decreasing the overall makespan of the application and increasing the sleep time. In addition to power saving, adaptivity helps to achieve fault recovery, as a faulty subsystem can be shut down and replaced by a backup subsystem. The proposed concept has been evaluated by an example scenario. The results exhibit that using the proposed concept, makespan of the processor and consequently the energy consumption are reduced. In addition to energy, the amount of the used memory for storing the communication schedules is also decreased.**

## I. Introduction

The area of Mixed-Criticality Systems (MCSs) [1] builds upon an increasing trend to integrate components with different levels of safety assurance onto a single shared platform. At the same time, these platforms are migrating from single core to multi-cores to gain the tremendous potential in terms of the computational capacity and energy efficiency. This calls for high-performance inter-core communication platforms that interconnect the cores and address the demanded safety assurance levels. This is driven further by the development of more powerful hardware and the demand from industrial sectors, such as aerospace, health-care and automotive, to save space, power, weight and certification costs.

Time-triggered communication was introduced to address determinism, which is highly demanded in safety-critical systems. Time-triggered systems make the design simple through implicit synchronization, which is offered by a global time base. In addition, the probability of a fault in time-triggered systems is lower, as the tasks are segregated and a failure of a faulty component does not interfere with other components [2]. The main drawback of using time-triggered systems is the absence of flexibility against varying resource demands. Resource requests in MCS systems can be highly dynamic and data dependent, which lead to overheads in terms of energy consumption. Hence, adaptivity in communication accommodates energy efficiency. Different techniques can be used to make the system energy efficient. In this paper, Dynamic Voltage and Frequency Scaling (DVFS) and *Clock Gating* will be discussed. DVFS reduces the energy consumption by controlling the voltage and/or frequency. By clock gating, the energy consumption is reduced through disabling a part of the circuit during the idle state. In addition to the above mentioned approaches, shortening the "make span" helps in saving energy. Normally, the injection time is determined based on the Worst-Case Execusion Time (WCET) of running application which is sending a message to another tile. However, if the design can detect when the running task is terminated, the underlying communication system can transfer the message to the destination earlier and the system can enter the standby mode, thereby less energy will be consumed. However, the computation of Worst Case Execution Times (WCETs) for multi-core processors is challenging by itself [3].

In addition to energy saving, reconfiguration enables the system to adapt with the availability of resources (e.g., power, scheduling, bandwidth). For instance, in a MCS, if the battery level goes below a certain level, unnecessary subsystems can be shut down (power-safe mode) to save the remaining power. When it comes to fault recovery, reconfiguration enables shutting down a faulty subsystem (e.g., a virtualized partition by a hypervisor or a core) based on the information received from the monitoring services. In this approach, the fault is detected and located to drive out the faulty node and run the task on other nodes to make system fault-tolerant.

Hence, efficient adaptation mechanisms are required to provide fault recovery and to make the system adaptive to the changes in the environment or resource requests by deploying different precomputed resource allocations, while keeping the system at a safe state. Due to safety concerns, reconfiguration of safety-critical systems is often reduced to selecting system-wide modes out of statically defined scheduling tables, to have a priori knowledge of the permitted component behavior, as required in such systems.

A system can face up with different types of events, which are categorized in two main groups: volatile and non-volatile events. The utility of volatile events is lost quickly and the knowledge about these events is not useful after expiration (e.g., slack). Therefore, the adaptivity of system for this type of events is supported within the period. The other group of events is called non-volatile events. These events are slow-paced and stay in the system longer than one period (e.g., battery, temperature). In this paper, three types of events are considered.

*Slack* time is defined as the difference between the WCET and the real execution time of computational tasks. The slack event occurs if the task is finished before the WCET. A *Battery* event is a specified energy percentage, at which the system goes to the low-power mode to extend the system lifetime. A *fault* event is used to represent a fault occurrence in the system.

978-1-5386-7656-1/18/31.00

For example, by fault detection and localization, a faulty core can be removed from the system and tasks are migrated to another core.

This paper proposes adaptive communication services for a safety-critical system in the presence of different types of events, making the system energy-efficient and fault-tolerant. The proposed solution uses adaptive communication through changing the schedule of the system based on detected events. Each event has a dedicated schedule, which is applied at run-time. The novelty of this paper is using the compressed data for schedules instead of saving the whole schedules.

This paper supports deterministic and dynamic routing algorithms and also makes the system more fault tolerant and energy efficient. The proposed concept can be applied to different safety-critical industrial and medical domains, such as avionic, railway, automotive and health care systems. In addition, it supports both dynamic and static routing.

This paper is structured as follows. Section II focuses on related work and represents similarities and differences of the proposed solution with already existing low-power techniques. Section III introduces the architecture and functionality of adaptive time-triggered MPSoC with more details about the NoC adaptation unit, while section IV presents the functionality of the proposed solution with an example scenario. In section V, the experiment results and the evaluation of the proposed approach are described. Section VI provides a conclusion and future work.

## II. Related Work

Adaptive communication as well as time-triggered communication have been addressed in previous works. However, adaptivity in time-triggered on-chip communication is not yet solved.

The Time-Triggered Network-on-Chip (TTNoC) [4] introduces an on-chip time-triggered interconnect with simple routers for predictable communication in real-time systems. This Network-on-Chip (NoC) with a pseudo-static communication schedule allows for a high bandwidth interconnect with inherent fault isolation for heterogeneous components of possibly different criticalities. In this architecture, adaptivity has been introduced by in a very restricted extent.

The TTNoC architecture supports multiple periods for the periodic transmission of messages. Subperiods are a fraction of the main period and can be deactivated individually by the application layer. However, in this solution, there is no possibility for shifting the injection time, but only the activation and deactivation of the injections which belong to a period is possible.

The European DREAMS project introduced a NoC architecture [5], [6] for mixed-criticality systems, which supports reconfiguration for a set of parameters. The DREAMS NoC (DRNoC) provides the adaptability based on the application needs (e.g., change to the fail-operational mode), environment changes (e.g., significant change in the temperature) and platform feedbacks (e.g., low battery). This provides energy efficiency (e.g., by shutting down selected entities) or fault-tolerance (e.g., by blocking the communication channels of the faulty entities). However, in the DREAMS solution, it is not foreseen to reconfigure the on-chip communication in each communication cycle and it is comparable with the non-volatile type of reconfiguration, which is introduced in this work.

As described above, the concept of adaptability has been addressed in prior research, but in a different context. In both approaches, support for adaptivity has been introduced to achieve fault-tolerance, different system operational modes and energy efficiency. However, in the proposed approach in this work, adaptivity is supported immediately within the ongoing operational cycle, as well as in subsequent operational cycles. The benefit of the presented solution is that adaptation of the chip-level services can take place in a significantly higher resolution, even during a single operational period of the periodic activities.

## III. Adaptive Time-Triggered MPSoC

The overall architecture of the proposed adaptive time-triggered MPSoC is illustrated in Figure 1. In this architecture, the MPSoC is composed of tiles that are interconnected by a NoC.

Each tile may contain a multi-core processor and is connected to the interconnect by a Network Interface (NI). The processor core can run its own operating system or a bare-metal application. In the NoC design, each tile is connected to one router, which is used for transferring the data from a source tile to a destination tile.

In this paper, the source-based control approach is described to keep the example understandable. In source-based routing, the route through which the message traverses is determined at design time by definition of the destination address of the messages. In addition, it requires simple routers and also offers determinism, which is essential for a safety-critical system.

In the proposed architecture, in addition to the processor cores and NI, each tile contains the following hardware building blocks in order to support adaptive communication.

**Context monitor unit**: The current system status in each tile needs to be monitored to be able to adapt the behavior of the system in an adaptive time triggered MPSoC. The context monitor unit observes the cores and the running software on the cores of the same tile locally. This observation, which enables the dynamic reaction of hypervisor or operating system to an event is stored at the dedicated ports of each tile. Ports provide interface between the core and the NoC and store the messages until they are delivered to the NoC [5] The observed data that is stored at the ports is called *local context*. This unit interacts with the cores and the context agreement unit in each tile. It receives the local monitored behavior (e.g., task finish time) from the cores and sends it to the context agreement unit.

**Context agreement**: The main motivation of this unit is to provide the system stability by establishing a chip-wide consistent global state. The context agreement unit is responsible for collecting and exchanging the local monitored contexts, which are used for schedule changes between all the tiles. This unit determines the next state of the system based on the current state. The current state of the system is identified by the global context, in conjunction with the events. The context agreement unit interplays with context monitor unit and the NoC adaptation unit at the core level in each tile. This unit receives the local monitored context, makes an agreement

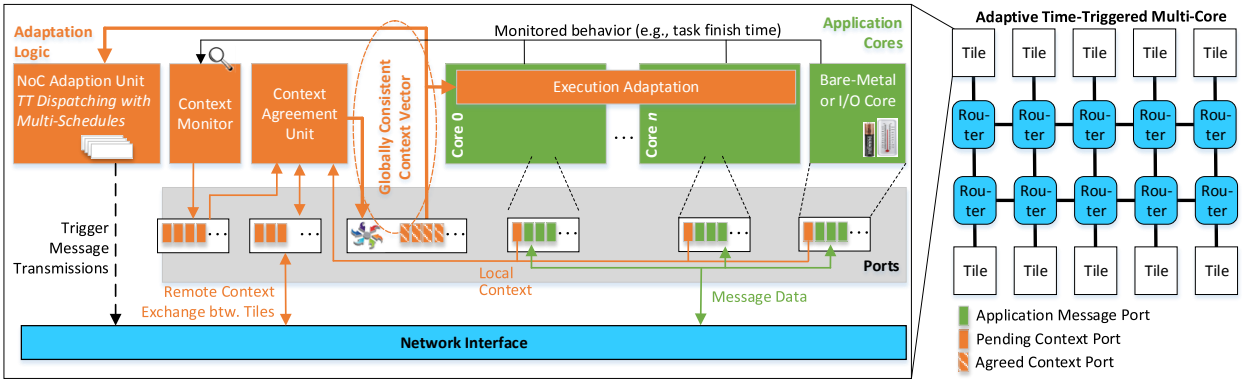



Fig. 1. Overall Architecture of Adaptive Time-Triggered Communication

for all the context agreement units and reports the Globally Consistent Context Vector (GCCV) as an output to the NoC adaptation unit. The GCCV contains the information of all the events of the system. There are two approaches to exchange the local context between the tiles: NoC-base approach and FIFO approach [7]. The proposed concept applies the FIFO approach, which is using a dedicated link between network interfaces is used for the propagation of the local context. The propagation is performed in a ring where the context agreement unit sends/receives the local context to/from the network interface.

**NoC adaptation unit**: The main challenge to have an adaptive time-triggered communication is consistency and robustness of switching between schedules. The NoC adaptation unit provides this consistency through aligning schedule changes. This unit provides the adaptation of schedules and triggers message transmissions with the adapted schedule. It fetches the GCCV at a predefined time to change the schedule at run-time. After changing the schedule, the injection of the messages is triggered by the selected schedule. The schedules are stored as a linked list structure, which is predefined at design time. Each state in the system corresponds to a unique schedule. The generation of linked list schedule is explained in the next section. NoC adaptation unit interplays with the agreement unit and NI by receiving the globally consistent context vector from the context agreement unit and triggering the message transmission through the NI.

**Execution adaptation**: This unit supports the adaptation of the execution resources which provides the consistent switching between schedules for operating systems.

### A. Linked List Multi-Schedule Generation

The Linked List Multi Schedule (LLMS) is a schedule storage, which is used to optimize the memory usage in the adaptive system. To generate the LLMS for the adaptive time-triggered system the following three tools are used: (a) Meta-Scheduler, (b) Delta Graph Generator and (c) LLMS Generator. Figure 2 represent the steps to generate the LLMS.

**Meta-Scheduler:** In this paper, a meta-scheduler tool is used to generate adaptive schedules [8]. Adaptive schedules are used to adapt the system according to the runtime events, which are important for time-triggered scheduling within the MPSoC. Adapting the system is handled by switching between different schedules based on the occurrence of the events in the system.

The Meta-scheduler deploys a Platform Model (PM), Application Model (AM) and Context Model (CM) to create the adaptive schedules. The AM contains the number of tasks, the WCET of each task and the precedence constraints [9]. The PM contains some information about the physical platform like the number of tiles and links between tiles. The CM defines event types and event execution time.

The CM is an important part of adaptive schedules , because it specifies the conditions for generating schedules. In this model, system designers define all the events that occur at run-time. Fault, slack, battery and application modes are supported by the proposed solution to improve reliability and energy efficiency of the system. In this paper, events are categorized depending on the urgency of their processing into *volatile* and *non-volatile* events. Volatile events are those events that must be monitored by the platform within the period and the platform is adapted immediately. Immediate fault recovery is an example for volatile events, as the correct operation of the system must be preserved. Another example is slack, which is considered as a volatile event. Hence the appearance of slack is checked at well-defined points in time during the execution of a time-trigged task in each period. In contrary, non-volatile events are those events that do not require an immediate monitoring, processing and reaction of the system, but they could be handled during the next period. Battery and application modes are non-volatile events and the reaction to those events is less urgent. In case of fault-recovery, if the purpose is to reestablish the redundancy degree and be prepared for another fault, it will be a non-volatile event. In this paper, the *Fault* refers to non-volatile fault.

**Delta Graph Generator:** Depending on the number of events, the meta-scheduler can provide hundreds or even thousands of adaptive schedules, which requires a large memory space. This memory usage can be optimized by storing just the differences between schedules instead of the whole schedules. The optimized schedule is called *multi schedules graph*, which is provided by the *Delta Graph Generator*. Each branch in the graph represents a new adaptive schedule. On the other hand, the decision making process must start in a well-defined time. In this way, in order to switch to a new state based on the most updated status of the system, the NoC adaptation unit

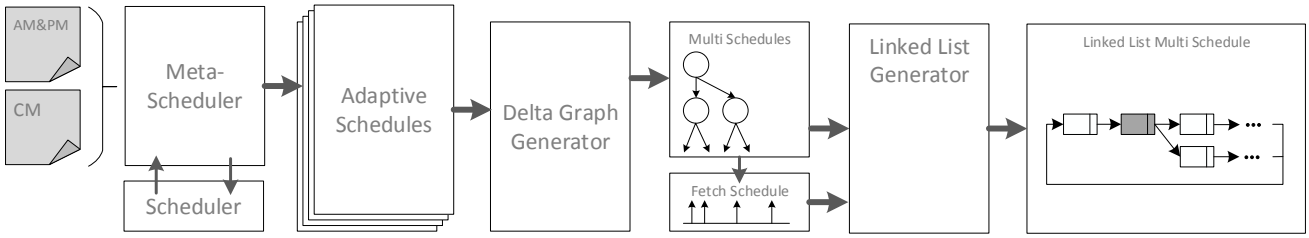



Fig. 2. Generation of Linked List Multi Schedule

needs to fetch the GCCV at the latest possible time, which is known as *Branching Point* time. The branching points are extracted from the multi schedule graph and calculated based on the required time for schedule changing.

This branching point time is calculated by subtracting the time that is required by the hardware for changing the branch (which can be calculated at design time) from the injection instant (which is given by the schedule). A set of branching times is known as *fetch schedule* and needs to be merged into the multi schedules.

**LLMS Generator:** The LLMS is generated using the multi schedule graph and the fetch schedule. In the first step, the LLMS generator converts the multi schedule graph to a linked list. Then the fetch schedule is merged into the linked list. The result is an LLMS which is compressed schedule information and suitable for adaptive communication.

### *B. Linked List Multi Schedule Structure*

The LLMS is defined individually for each tile. Conceptually, the LLMS is made up of injection entries, which are connected in a circular linked multi-list data structure.

The entry types are categorized as Message Transmission (MT) and Branching Point (BP). Depending on the entry type, the information which is stored for the entry is different. Each MT entry stores the *PortID*, which represents the ID of the port to be dequeued, *Instant*, which represents the phase of the injection and *Next* which contains the memory address of the next injection entry. The BP entries contains *instant* which represent the time to fetch the *GCCV*, the *NextTaken* and *NextNotTaken*, which refer to the next addresses. The *GCCV* contains the information for all the events and each tile requires a portion of the information. Therefore, the BP entries contain a binary mask known as *BitMask* to extract the event occurrence for each tile individually.

### *C. NoC Adaptation Unit*

As mentioned in the introduction, one of the main challenges for adaptation is robustness and consistent switching to the new state, which is provided at run time by aligned schedule changes. The NoC adaptation unit provides this prerequisite for a consistent new state. Figure 3 represents the architecture of this unit, which consists of three different blocks: context register, LLMS and adaptation manager.

*Context register* is defined to store the GCCV, which is received from the context agreement unit. The stored GCCV is fetched at a dedicated time, which is defined by the LLMS.

As described above, the LLMS contains the different schedule and controls the timely injection of the messages from the

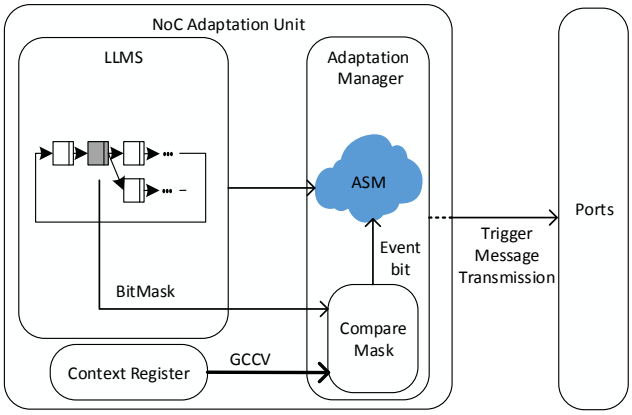



Fig. 3. Adaptation Manager

port to the NoC. This block is aware of the timing and uses a low global time base as a low-frequency digital clock for a consistent schedule change. Therefore the transmission of the message are triggered based on the time that is defined by the linked list entities. The switching between different entities in the linked list is based on the stored GCCV.

The *adaptation manager* traces the GCCV and declares the new schedule based on the GCCV and the BitMask. This building block contains a Compare Mask unit and an Adaptation State Machine (ASM). The ASM operation is presented in Figure 4. This state machine is triggered at the instant of time, which is defined for each entity of the LLMS. Depending on the entity type, the next state is selected. For an MT type entity, the PortID is read, the message in injected to the NoC at the respective instant of time and the next points to the address of the next entity. In case of a BP type entity, the compare mask fetches the GCCV and reads the BitMask to define the occurrence of the specified event. If the event is occurred the T_Entry (NextTaken) is selected as the next entry. Otherwise, the NT_Entry (NextNotTaken) schedule is selected. After the last entity, next points to the first entity of the next period. This process is repeated for each period.

## IV. Example Scenario

The functionality of the NoC adaptation unit is simulated and validated by an example scenario. This example employs four tiles which are connected by a NoC in a 2 x 2 mesh topology. Three tiles are used for the normal application and one tile is used as a backup tile in case of a fault event. This example represents adaptability in case of slack as a volatile event and a fault as a non-volatile event.

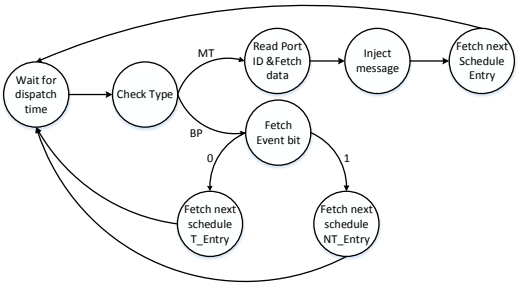



Fig. 4. Adaptive State Machine

An application scenario which is defined based on an avionic use-case is used to validate the proposed NoC adaptation unit. This avionic application contains tasks with four different phases:

1) Reading all the sensors and remote data
2) Processing sensor and remote data
3) Controlling the flight
4) Setting the motors

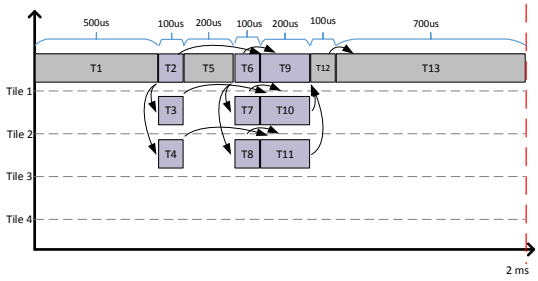
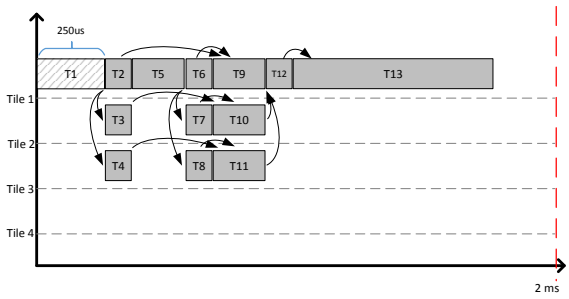


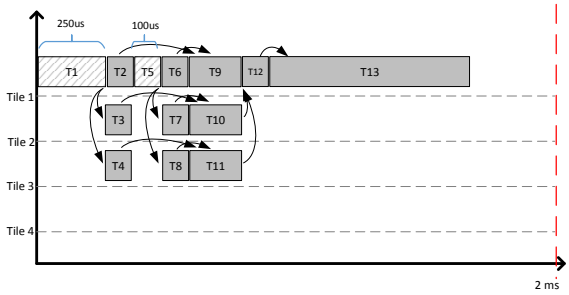

Fig. 5. WCET and Communication Channels

For making the system fault tolerant, a new task's phase is added to the application realizing Triple Modular Redundancy (TMR) for fault masking. In this example, TMR is used for critical tasks which are processing sensor data, processing remote data and flight control. These applications are running on three different tiles and the result is sent to the voting task. The voting task combines the results of the three flight controlling processes. This example supports one fault per period which is masked by TMR.Afterward, the faulty tile is dropped from the system and the backup tile is applied to reestablish the redundancy and prepare for the next fault.

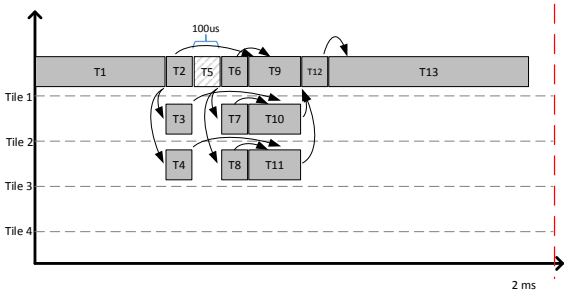
Figure 5 represents the WCET of the tasks and the communication between the tasks on the different tiles. The arrows show the communication channels between the tiles in which the period is considered as 2ms. The tasks are running each 2ms unless there is an event in the system. In the proposed example, each tile has one processing core and tasks are dedicated to each core. Tile 1, Tile 2 and Tile 3 run the same tasks for processing sensor data, processing remote data and flight controlling and the result is sent to the voting task on tile 1. These critical tasks are represented in purple color. In case of a fault on Tile 1, Tile 2 or Tile 3, the faulty node is detected by the voting task. If the voter detects a fault, the correspondent tile is dropped and the backup tile which contains the related task is used instead to reestablish the redundancy.



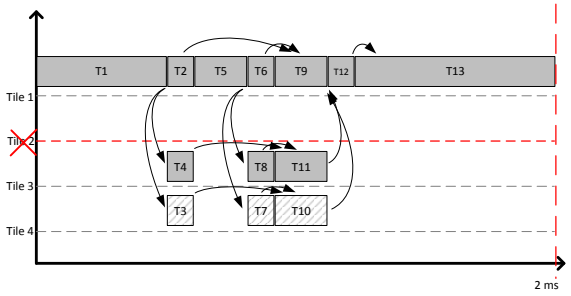

(a) Event 1: Slack on Task1




(b) Event 2: Slack on Task 1 and Task 5




(c) Event 3: Slack on Task 5




(d) Event 4: Fault on Tile 2

Fig. 6. Schedules with Different Events

The following types of events are considered in this example:

- **Volatile Event:** Slack on Task 1 and Task 5.
- **Non-volatile Event:** Fault on Tile 2.

The existence of slack in the tasks is checked after 50% of WCET. In case of slack occurrence the schedule is changed. The result is a shorter makespan, which is used to make the system more energy-efficient. In slack events, the communication channel injection time change and the messages are send earlier.

In this scenario, the fault can be detected and located with the majority voting in tile one. The faulty tile is excluded

from the system and the tasks of the faulty tile are running on spare tile. In general, the number of spare tiles depends on the number of faults and available resources. In this example, one spare tile is used to handle the tasks of the others tiles. After fault events, the destination of messages can be different from the original basic schedules. Therefore, the schedule changes also involves a modification of the communication channels.

Figure 6 represents four types of event scenario: schedule with slack event on Task 1, schedule with slack event on Task 1 and Task 5, Schedule with slack on Task 5 and schedule for fault event on Tile 2. The modification in the tasks is presented by hachure. In Figure 6(a), the slack is considered for Task 1 which initially has the WCET of 500 us. Since the slack occurrence is checked after 50% of the WCET, the message is sent after 250 us. Figure 6(b) shows the slack events on task 1 and Task 2 in one period, which save 250 us of Task 1 and 100 us of task 5. In case of slack happening just on Task 5 which is shown on Figure 6(c), the makespan is saved by 100 us. Figure 6(d) is representing the fault event on tile 2. In this case, tasks 3,7 and 10 are run on the tile 4 and the communication channels of tile 2 with the others are not valid any more. Tile 4 also contains the task of tile 1 which are deactivated and is activated in case of fault occurrence on tile 1.

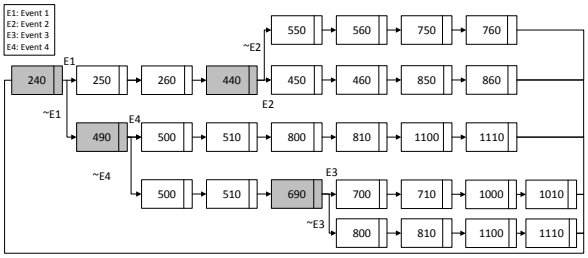



Fig. 7. LLMS Format

The linked list representation of the schedule for different slack events based on schedules of Figure 6 is represented by Figure IV. The BP entries are defined by the gray color and the MT entries with white color. As described in the linked list multi schedules structure, the instant of time for the BP entry is calculated based on the fetch time schedule. The value on each branch represents the occurrence of the specified event.

## V. Experimental Result

In this experiment, the prototype is implemented on a Xilinx ZC706 ZYNQ-7000 FPGA-based board. The architecture has been implemented using a 2x2 Mesh network, in which the NIs are interconnected by an ordinary event-triggered NoC. For simplicity, source-based routing has been deployed. The results represent the benefit of the proposed architecture and are shown in the following two aspects:

**Energy Efficiency:** Energy efficiency is the main advantage of using the proposed concept. Hence, the evaluation contains slack and fault events to exhibit the energy efficiency. The slack event provides shorter makespan, thereby reducing the energy consumption using the clock gating technique.

The maximum energy saving occurs when all the tasks have a slack.

In case of fault, the energy efficiency is the same as before, as the hardware usage does not change. Table I represents the energy saving of the example scenario in Figure 6 for different events. The results show that the maximum energy saving happens in event 2, since makespan is reduced by 17.5% and the slack happens after 50% of T1 and T5. This reduction value can be different based on the number of tasks which are selected for the slack event and the execution time of the tasks. If the slack occurs at 50% of the execution time for all the tasks, the makespan will be reduce by 50% in the best case.

| Event Number | Base_Makespan | makesapan | Energy Saving% |
|---|---|---|---|
| Event 1 | 2 ms | 1,75 ms | 12.5 |
| Event 2 | 2 ms | 1.65 ms | 17.5 |
| Event 3 | 2 ms | 1.9 ms | 5 |
| Event 4 | 2 ms | 2 ms | 0 |

TABLE I
Energy saving for different events

**Memory Storage:** The proposed concept improves the required memory for the storage of the time-triggered schedules, as only the differences are stored. Figure 8 represents the memory optimization of the proposed architecture with considering 2 slack events in the system. In this figure, the Y axis represents the amount of memory usage and the X axis represents number of tasks. This figure compares the memory usage for storing the schedule for adaptive schedules with the average LLMS size and minimum LLMS size. The result shows that the memory is optimized more than 50% in best case.

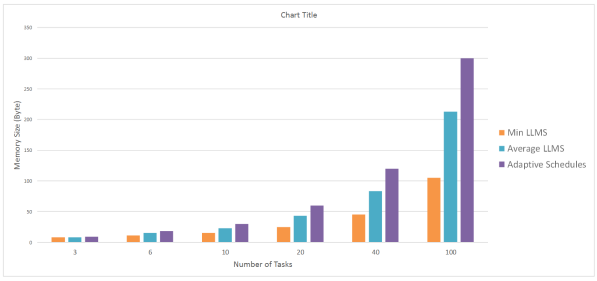



Fig. 8. Memory Optimization

## VI. Conclusion and Future Work

This paper presents an architecture that provides adaptive communication services for a time-triggered system in presence of different types of events. In addition, the proposed concept provides energy efficiency and fault recovery through adaptive behavior to the events. The adaptivity is supported by changing the schedule of the system at run-time. The result shows the improvement of schedule storage and reduction of energy consumption. The design is simulated and implemented by a ZC706 FPGA-Board. In the future, the measurement of dynamic power for each state is planned to be carried out. This will be technically done by adding the power inception library to our application. In addition, we will continue with adding a self fault recovery module to the proposed architecture.

## Acknowledgements

This work has been supported by the European project SAFEPOWER under the Grant Agreement No. 6872902.

## REFERENCES


[1] Editted by: H. Ahmadian, R. Obermaisser, J. Perez, *Distributed Real-Time Architecture for Mixed-Criticality Systems*, 1st ed. USA: CRC Press, 2018.

[2] Editted by: R. Obermaisser, *Time-Triggered Communication*, ser. Embedded Systems. USA: CRC Press, 2012.

[3] "Position paper – multi-core processors, CAST-32A," Certification Authorities Software Team (CAST), Tech. Rep., 2016.

[4] A. Wasicek, C. El Salloum, and H. Kopetz, "A system-on-a-chip platform for mixed-criticality applications," in *Proc. of IEEE Int. Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, 2010, pp. 210–216.

[5] H. Ahmadian and R. Obermaisser, "Time-triggered extension layer for on-chip network interfaces in mixed-criticality systems," in *Digital System Design DSD 2015*.

[6] H. Ahmadian, R. Obermaisser, and M. Abuteir, "Time-triggered and rate-constrained on-chip communication in mixed-criticality systems," in *Proceedings of 10th International IEEE Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC16)*, 2016, pp. 117–124.

[7] A. Lenz and R. Obermaisser, "Global adaptation controlled by an interactive consistency protocol," *Journal of Low Power Electronics and Applications*, vol. 7, no. 2, 2017.

[8] B. Sorkhpour, A. Murshed, and R. Obermaisser, "Meta-scheduling techniques for energy-efficient robust and adaptive time-triggered systems," in *Proc. of 4th IEEE International Conference on Knowledge-Based Engineering and Innovation (KBEI)*. IEEE, 2017, pp. 143–150.

[9] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Proc. of 28th IEEE International Real-Time Systems Symposium*, Dec 2007, pp. 239–243.