Fault Recovery and Adaptation in Time-Triggered Networks-on-Chips for Mixed-Criticality Systems

Hamidreza Ahmadian, Farzad Nekouei and Roman Obermaisser
Chair for Embedded Systems
University of Siegen
Siegen, Germany

Abstract—Adaptivity in terms of fault recovery and energy efficiency alongside with mixed-criticality support are demanded in today’s embedded systems. Safety-critical systems are desired to switch between precomputed resource allocations at runtime based on the monitored information from the platform. In addition, those systems are desired to adjust their internal behavior with regard to a change in the environment, while operating at a desired safety level. At the same time, resource requests in such systems can be highly dynamic and data dependent. Aiming at meeting a superset of all worst case demands leads to unaffordable overheads in terms of resource utilization. Hence, efficient resource management mechanisms are required to provide fault recovery and to make the system adaptive to the changes in the environmental or the resource requests, while keeping the system at a safe state. This paper introduces a solution for supporting resource management in networks-on-chips that fulfills the requirements of adaptive mixed-criticality systems and proposes an architecture that establishes fault recovery by switching between precomputed resource allocations based on the statistical and diagnostic information.

I. INTRODUCTION

The area of Mixed-Criticality Systems (MCSs) builds upon an increasing trend to integrate components with different levels of safety assurance onto a single shared platform. 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. Such systems must seek to address challenges, such as ensuring safety by isolation of high integrity tasks, at the same time with an efficient resource utilization. In addition, managing the available resources at runtime provides fault-recovery which increases the reliability and thus is desired in MCSs.

Safety-critical embedded systems are desired to adjust their internal strategies with regard to a change in the environment while keeping the system in a safe state. For instance, aircraft flights progress through phases (e.g., taxiing, take-off, climbing, level flight, etc.) and automotive systems have modes to cover start-up, cruise control, driver control, ‘limp home’ etc. At the same time, resource requests in MCS systems can be highly dynamic and data dependent and aiming at a superset of resource demands in all environmental situations leads to unaffordable overheads in terms of resource utilization. In addition, supporting 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. Hence, efficient resource management mechanisms are required to provide fault recovery and to make the system adaptive to the changes in the environmental or the 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.

Fault recovery can be considered in different aspects, i.e., distributed or centralized, on-chip, off-chip or at hypervisor layer. In mixed-criticality network-on-chips, fault-recovery should be addressed in the use of both the computational resources (e.g., the allocation of tasks to the processors) as well as the communication resources (i.e., the use of communication network for interaction between the processors). In addition, as in mixed-criticality systems applications of different safety assurance levels coexist and interact on a single shared platform, a combination of both above mentioned challenges should be addressed. The recovery strategies are relevant for safety-critical applications and adaptive Quality of Service (QoS) will be used in low-critical applications.

Mode-switching in safety-critical systems has been the subject of considerable study over a number of years [1] (e.g., cf. mode change protocols [2]). Reconfiguration support has been addressed as a challenge in future avionic architectures [3] and has been addressed by moving applications hosted on a faulty computing module to a spare computing modules in [4]–[6]. In [7], a solution for the fault recovery at the hypervisor layer in the avionic domain has been introduced. Permanent core failures and temporal core overload situations are detected and recovered by moving the application components to another partition, thanks to the hypervisor.

Adaptivity in low-critical communication infrastructures has been addressed by the adaptive QoS to help managing packet loss, delay and jitter in use-cases like voice and video streaming over the network. In [8] and [9], runtime adaptive techniques for meeting the desired QoS have been addressed.

This research investigates how the requirements of adaptive mixed-criticality systems are fulfilled by using the proposed solution in conjunction with the resource management solutions at other layers (e.g., DREAMS resource management introduced in [7]). We show how the proposed approach...
establishes fault-recovery and efficient resource utilization in Mixed-Criticality Network-on-Chips (MCNoCs) by monitoring the resource requests and reconfiguring them based on a priori known recovery strategies. The proposed architecture performs the reconfiguration of the Time-Triggered Extension Layer (TTTEL) [10], [11] and provides a solution for the local monitor and the local scheduler that are introduced in [12]. These interfaces serve a resource manager that switches the system mode, based on the information provided by the local monitor. The local scheduler allocates the resource, based on the new chosen mode.

The remainder of the paper is structured as follows. In Section II the requirements an adaptive on-chip communication infrastructure in mixed-criticality systems are elaborated. State-of-the-art solutions are analyzed in Section III. Section IV explains how the system model should look like and in Section V, the proposed architecture and the different building blocks are depicted. Section VI presents a test scenario to evaluate the proposed architecture and Section VII concludes the work.

II. REQUIREMENTS OF ADAPTIVE MIXED-CRITICALITY SYSTEMS

There is an increasing trend towards the development of more powerful hardware in mixed-criticality embedded systems, to afford more performance. This is driven further by the demand from industrial sectors, such as aerospace, health-care and automotive, to save space, power, weight and certification costs. This leads to a migration to multi-core or many-core System-on-Chips (SoCs), in which a MCNoC plays an important role. In such systems, major challenges are safety as well as efficient resource utilization, which seem contradictory at the first glance. This section provides a brief look at the requirements as well as basic, but essential services of such Network-on-Chips (NoCs).

A. Requirements of Mixed-Criticality Systems

The integration of subsystems with different criticality levels requires isolation which prevents low-critical subsystems to interfere with those of higher criticality, thus eliminating the potential of failures in safety-critical subsystems caused by low-critical ones [13]. Otherwise, the entire system would have to be certified to the highest level of criticality, which is economically and technically infeasible for low-critical subsystems. Furthermore, bounded interference between low-critical subsystems is also desired in a state-of-the-art MCNoC to reduce development and validation efforts.

Fault containment is technically achieved by Temporal and Spatial Partitioning (TSP). Temporal partitioning provides temporal segregation of resource usage to guarantee that timing of a resource (e.g., duration and speed of the communication services) is not affected by other components, thereby avoiding the case, in which a message is delayed or does not arrive at the destination, due to an interference. Spatial partitioning provides data integrity by a separation of address spaces (at memory level) and defining the sender IDs (at network level). In this way, one sender cannot send a message with the ID of another sender, thereby, there will be no interference of messages.

Time-triggered systems inherently support TSP, thereby achieving fault-containment, in addition to predictability, determinism and inherent fault isolation. However, many mixed-criticality systems are desired to support the event-triggered communication for the low-critical application subsystems, to achieve an efficient bandwidth utilization and to provide legacy-application support. Combining the time-triggered and event-triggered paradigms requires segregation mechanisms which is elaborated thoroughly in [10], [11].

B. Requirements of a Reconfigurable Architecture

Support for reconfigurability in MCSs implies a considerable design, implementation and certification overheads. A number of requirements shall be fulfilled, in order to not invalidate the already existing characteristics of the system. For instance, any changes in the resource allocation should be a priori known, in order to avoid unknown system states. Reconfiguration of a low-critical subsystem shall not interfere with the operation of a safety-critical subsystem. The following, summarizes the requirements of a reconfigurable system.

a) Safe Reconfiguration: Reconfiguration activities should not disrupt the behavior of subsystems that are not subject to the reconfiguration activities. In particular, the aftereffects of the reconfiguration to the critical services shall be known beforehand.

b) Predictable Reconfiguration: The primary operation of the system after the reconfiguration shall not be corrupted. This challenge is also known as the problem of assured reconfiguration.

c) Bounded Time for Reconfiguration: The dynamics of the environment and the system requirements determine the available time for the new configuration. Hence, the time needed to adopt a new configuration shall be bounded.

d) Consistent Switch to new Configuration: It is necessary to provide a way to consistently switch between different configurations and operational modes. Intermediate configurations need to be avoided where subsets of nodes are in the new and others are in the old configuration.

e) Continuity of Service after the Reconfiguration: Resource requirements should be satisfied when the reconfiguration is terminated.

f) Robust Reconfiguration Mechanisms: The reconfiguration mechanisms are critical in the sense that a failure of the reconfiguration process has the potential of causing a global system failure.

g) State Preservation: Relevant internal state must be preserved while switching between configurations is in progress. For instance, the stored but not yet delivered messages shall be preserved if these messages are still valid in the new configuration.

III. RELATED WORK

Resource management is a concept that can be considered at different layers (e.g., operating system, hypervisor layer, off-chip network and on-chip communication network) and has been thoroughly dealt with by a number of researchers and projects in the past. In the MATRIX [12] project, a framework

---

1 Which is beyond the scope of this research.
for applying real-time off-chip resource management methods in video streaming of heterogeneous devices has been introduced. In the ACTORS project [14] (which is inspired by MATRIX), a resource manager has been developed that collaborates with a Linux scheduler. This approach provides support for hard constant bandwidth server reservations to adapt applications to changes in resource availability and to adapt the resource allocation to changes in application requirements. In the Spring project [15], Stankovik et al. present the Spring scheduling co-processor that improves the task and resource management services in real-time operating systems.

In the use-cases, resource management has been addressed in the ASAAC [4] project for military aircrafts and in Diana [5] and Scarlett [6] for the civil domain. In the DREAMS [7] project, a reconfiguration approach has been elaborated that addresses the reconfiguration strategies to deal with the failures and to bring the system back to a safe functioning state. Two types of failures have been addressed, a permanent core failure and temporal overload situations. However, these strategies have been defined and implemented at the hypervisor layer by software.

Dynamic reconfiguration at the chip-level interconnect has been addressed to achieve adaptive QoS. Motakis et al. [8] demonstrate a software solution that enhances dynamic adaptation of the NoC resources at the run-time. Ruaro at al. in [9] introduce monitoring and adaptation management which provides adaptive routing, flow priorities and mode switching for the monitored applications. This approach is however performed at the operating system level and implemented by software.

In the IDAMC platform [16], a monitoring and control mechanism has been introduced. The main purpose is to isolate a possibly faulty low-critical application/tile to still be able to guarantee the timing of safety-critical applications running on other tiles. The monitoring mechanism in the network interface collects information about the usage of shared resources by recording the resource accesses and detects the violations by comparing with the maximum allowed value. In case of a violation, the faulty tile can be reset, disabled, paused or a message will be sent to the system controller. As the IDAMC platform does not support the transmission of time-triggered and rate-constrained messages, in the provided monitoring and control solution, there will be no support for monitoring or reconfiguration of the temporal properties of those messages.

The proposed approach is inspired by the concept that has been introduced in MATRIX and evolved in ACTORS. Moreover, similar to the DREAMS approach, fault-recovery serves as the major motivation for supporting the reconfiguration. As described above, the mentioned approaches are only applicable at the processor level or off-chip communication level and have been implemented by software using the operating system or the hypervisor (e.g., XtratuM). In contrast, in the proposed approach, the reconfiguration is applied at the chip-level communication and the mentioned requirements in Section II has been fulfilled. For instance, as the proposed architecture is implemented by hardware, a predictable approach in terms of the reconfiguration time and behavior can be achieved.

IV. System Model

An example SoC in mixed-criticality systems that contains several tiles interconnected by on-chip routers and physical links, is depicted in Figure 1. Tiles act as a source or a sink in the message-based communication. Each tile can contain one or more processor cores and run its own operating system or a bare-metal application. Alternatively, it can be virtualized by a hypervisor [17]. Hypervisors are widely used in mixed-criticality systems in order to provide components of different criticality levels with isolated execution environments [18] (dashed boxes on top of the hypervisor in Figure 1). In addition, system-specific tiles are supported as such a DDR controller, an off-chip gateway or a peripheral controller.

Independent from the use-case, the following basic services are expected to be supported by a NoC in MCSs [10], [19], [20]:

- Communication Services: In MCNoCs, delivery of safety-critical message with minimal jitter and bounded worst-case delay as well as bounded interference between the low-critical messages is required.
- Support for Different Timing Models: Different application domains demand diverse timing models and consequently different types of communication. Time-triggered messages [21] offer predictability, determinism and inherent fault isolation. Best-effort messages require no resource reservation and no timing restrictions, thereby, providing legacy-application support and efficient bandwidth utilization. Rate-constrained messages [22] offer a reasonable trade-off between resource sharing and bounded worst-case latency by defining the rate-constraints as well as the priorities. More specifically, each rate-constrained message is relayed to the destination only if the Minimum Inter-arrival Time (MINT) is elapsed and there are no event-triggered messages of higher priority.
- Global Time Base [23]: In order to establish a chip-wide collision-free communication of safety-critical messages, the operation of different entities shall be harmonized by a global time base to have a common understanding of the time, despite different clock domains. Otherwise, two different messages might meet each other at the same router due to the clock drift at one of the sender nodes.
- Interface to the Application: Ports provide the interface to the NoC for the application layer. They are communication terminals of application components and decouple them from each other and from the NoC. Using the ports,
the design of the NoC can abstract from the application components and the implementation technology of the tiles.

- Encapsulated Communication Channels: The MCNoC shall preserve the isolation established by other layers (e.g., the hypervisor at the application layer or off-chip gateway) by establishing a one-to-one mapping between the ports and the software components at the hypervisor level (cf. Figure 2).

- Monitoring and Reconfiguration Services: Reconfiguration in MCNoCs makes the system fault-tolerant by avoiding fault propagation through changes to the system. The resource manager decides whether a change is required, or which mode shall be selected for the system based on the information provided by the monitoring services. The monitoring and reconfiguration services can be implemented on top of the communication services and can use ports to interact with other elements of the system.

A. Architecture of Reconfigurable Systems

Though dynamic reconfiguration has been addressed at different levels of abstractions, one can identify a uniform architecture independent of the use-case [7]. We define a reconfiguration domain (cf. Figure 3) as a subset of an adaptive system which can be reconfigured independently. Each domain is basically composed of several entities that constitute an effective mechanism for monitoring of available resources in the domain and (re)scheduling them based on the defined algorithm, which is identified based on the use-case. Over the entire system, several reconfiguration domains are harmonized by a single system manager, which considers global considerations and interacts with the resource managers.

For each domain there is exactly one resource manager, which receives the resource requests and reserves the resources within the domain [24]. Each time a resource variation occurs, the resource manager makes a new decision and adjusts the resource allocations accordingly. The resource manager shall have the knowledge about the availability of the domain resources in order to be able to allocate them. This information is obtained from the MONitoring interface (MON), which collects the diagnostic or statistical information of the resources. At each resource, there is a Local Resource Scheduler (LRS) that applies the obtained schedule from the resource manager.

Below, the operation of each of the mentioned building blocks is elaborated.

B. Monitoring Services

The MON provides two types of information, firstly, the STATistical information (STAT) and secondly, the DIAGnostic information (DIAG). STAT includes information which can be used by the resource manager or in some cases, by the application layer (e.g., number of available messages at a port, if a port is empty or full). DIAG information will be generated, only if any unexpected behavior is spotted at the resource (reading an empty port, writing into a full port). This information will aid the resource manager to detect faulty software or hardware element.

In the proposed architecture, three groups of information are monitored (including STAT and DIAG).

a) Monitored Artifacts at the Ports:

Ports are communication terminals of application components. They decouple the application components from each other and from the NoC. Therefore, it is crucial to have information about the interface. Two types of ports are identified [10]. State ports employ a buffer with update-in-place semantics for the data area that is overwritten whenever the new data becomes available from the core or the network. In case of an event port, messages are enqueued to a FIFO which is composed of several buffers, each of which contains an event message. Hence, in case of an event port, the following items can be monitored at the buffer level as well as the port level.

The following characteristics at the ports can be monitored:

- Empty (STAT): represents whether the entity (buffer or port) is empty.
- Full (STAT): represents whether the entity (buffer or port) is full.
- Number of Messages (STAT): represents number of available messages inside the port. These messages can be enqueued by the application layer (in case of an output port), or by the NoC (in case of an input port). This value is in fact the difference between the read and write pointers of the respective FIFO.
- Length of Message (STAT): as the TTEL supports variable length for the messages, this information helps the driver for reading the received message at an input port.

b) Monitored Driver Errors:

As the driver is implemented by the software, the following faulty behaviors can be observed and captured by the monitoring services:

- Port Overflow (DIAG): represents the case, in which the driver attempts to enqueue a message, while the port is already full.
• Buffer Overflow (DIAG): represents an write procedure, in which, the driver keeps enqueue-ing the message, though the buffer is full.
• Port Underflow (DIAG): represents the case, in which the driver attempts to read an empty port.
• Buffer Underflow (DIAG): represents an read procedure, in which the driver keeps dequeue-ing the message, though the buffer became empty.
• Uncompleted Read Operation (DIAG): represents the case, in which the act of reading into the port is not terminated correctly by the driver (the act of writing is terminated by asserting the terminate signal) [10].
• Uncompleted Write Operation (DIAG): represents the case, in which the act of reading from the port is not terminated by the driver (the act of reading is terminated if the message is fetched entirely and the port becomes empty) [10].
• Reconfiguration Errors (DIAG): represents the case, in which the resource manager asks for an unknown reconfiguration (e.g., wrong destination, wrong port ID etc.)
  c) Monitored NoC Errors: As the underlying NoC may not meet safety-critical requirements, the following faulty behaviors can be observed and captured by the monitoring services:
  • Missed Deadline (DIAG): independent from the implementation of the underlying NoC, the traversal-time of the messages, in case of no traffic shall be bounded. Hence, if the safety-critical messages arrive at the destination after the defined deadline, this deadline overrun will be captured and reported.
  • Misrouted Packets (DIAG): independent from the implementation of the underlying NoC, if a packet arrives at a wrong destination and violates the source-based routing, this will be captured and reported.

C. Reconfiguration Services

The aim of the dynamic reconfiguration is to provide 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).

Reconfiguration at the NoC level can be applied to the following resources:

a) Ports: ports contain a number of parameters, like the MINT values, the destination address or the paths in source-based NoCs that can be updated by the reconfiguration services at run-time. A list of possible reconfigurable parameters follows:
  • Enabling or Disabling a Port: disabling a port provides fault-tolerance by preventing a faulty subsystem to inject malicious messages and affecting other subsystems. Moreover, it could help in reducing the power consumption, as the underlying NoC will not be deployed for the transmission of messages, originating from the disabled port.
  • Changing the Source-Based Routes: in case of mode-change or fault-recovery, an application subsystem can be moved to another partition, tile or chip. Therefore, the communication channels that used to end at the moving application subsystem, shall be updated based on the new location and this implies an update to the stored path or destination of the respective port.
  b) Network Bandwidth:
  • Enabling or Disabling a Period: time-triggered systems operate based on a time-triggered schedule which is comprised of possibly several periods. Each of these periods can be switched off and the respective activities of that period will not be triggered. This provides energy efficiency and fault-containment, as the communication channel of the respective period will be disabled.
  • Changing the MINT Value: in case the design supports the transmission of Rate-Constrained (RC) messages, changing the MINT value shapes the Event-Triggered (ET) traffic and can be used in mode-change. In addition, it could help in improving the energy efficiency by decreasing the frequency of message transmission. The reconfiguration of MINT value can be compared by dynamic QoS in adaptive NoCs and optionally can be exposed to the application layer. However, in case of safe-critical ports, the new value will be checked by the interface.
  • Schedule Change: in case the design supports transmission of Time-Triggered (TT) messages, several scenarios (e.g., mode changes, energy efficiency) could require update in the TT schedule. Of course, this update shall be managed globally and thus involves the global system manager.

V. The Proposed Architecture

The proposed architecture performs the reconfiguration of the on-chip communication services and provides a solution for a local monitor and a local scheduler at the NoC level. Technically this is achieved by addition of a Resource Management Interface (RMI) to an already existing TTEL [10], [11]. The TTEL exhibits a modular architecture that establishes safety as well as efficient resource utilization by offering the TT and ET communication paradigms and can be used as an extension layer at the Network Interface (NI) of existing NoCs. The RMI performs the reconfiguration of the TTEL, based on the commands from the resource manager and serves it by collecting and sending the monitoring information.

The communication between the components of the resource management services is performed on top of the communication services. More precisely, the communication between the resource manager and the RMI is performed via the communication channels that are established by the TTEL, as follows:
  • $Ch_{REC}$: This communication channel is a state TT channel starting from the resource manager and ending at the TTEL by a REConfiguration Port (RECP) (read only by the RMI).
  • $Ch_{STAT}$: This communication channel is a state TT channel starting from the TTEL by a MONitoring Port (MONP) (written into only by the RMI) and ending at the resource manager.
  • $Ch_{DIAG}$: This communication channel is an event RC channel starting from the TTEL by an ERRor Port
A. Master and Slave TTELs

In case an SoC is composed of several tiles and consequently, several TTELs, one TTEL would reside on the same tile as the resource manager residing and the remaining ones on different tiles that communicate via a NoC. This requires the possibility of a remote communication between the resource manager and the RMIs that are located on different tiles.

This challenge has been addressed by introducing different types of TTELs in the context of the resource management services. A master TTEL resides at the same tile as the resource manager is located and contains mirror ports (ports with dashed borders in Figure 5) that are coupled by the RMI ports of the slave TTELs, that locate on different tiles (tiles at the lower side of Figure 5). The resource manager interacts only by the local mirror ports and the communication services of the NoC take over the communication between the mirror ports and the RMI of the slave TTELs.

B. Position of the RMI inside the TTEL

As shown in Figure 6, the RMI, is located at the central part of the TTEL and interacts with several interfaces within the ports, the scheduler and the dispatcher via the internal data buses.

Once a new REConfiguration (REC) command arrives at the RECP, the RMI fetches the message from the port and extracts the command (e.g., enabling an application port or updating the MINT value at a port) as well as the payload (e.g., whether enable or disable or the new mint_value). In addition, at a configurable frequency, the RMI reads the status registers of individual ports and collects all information and enqueues the information into the MONP. Once an error happens at a port (e.g., port/buffer overflow or underflow), the RMI reads the error from the respective port and sends a message, which contains the port_id and the error_id to the resource manager using the dedicated ERRP. This message terminates with the value of the global time base as a time-stamp.

C. Structure of the RMI

As shown in Figure 4, the RMI contains three internal units, each of which operates independently. Below the operation of each unit is elaborated.

a) Statistics Unit: Each port (as shown at the bottom of Figure 4) contains a port status interface which captures the status flags, embeds them into a message and sends the message to the RMI, once its ID is declared by this unit. After collecting the statistics from all ports, this unit generates a long message, which contains all information appended by the value of the global time base and writes it into the MONP to be read by the resource manager.

b) Diagnosis Unit: Each port (as shown at the bottom of Figure 4) contains a port diagnosis interface which detects the faults, embeds them into a message and raises an error flag to inform this unit about an error. Afterwards, the diagnosis unit inside the RMI checks the ports based on the asserted flag and declares the ID of the port, so that the respective port diagnosis interface puts the error data on the bus. Afterwards, this unit writes the error message appended by the global time base value into the ERRP to be read by the resource manager.

c) Reconfiguration Unit: Once the resource manager writes a reconfiguration command into the RECP, this unit is notified and reads the message from the port. After extracting the ID of the port, the ID of the artifact which is to be reconfigured and the new value from the message, the reconfiguration unit declares the port ID and puts the reconfiguration message on the shared bus. Afterwards, the reconfiguration interface inside the respective port takes the message and applies the requested update.
VI. VALIDATION

This section provides the proof of concept and addresses a validation scenario which has been simulated, synthesized and implemented on a Xilinx ZYNQ-7000 FPGA. In this scenario, we tried to generate a fault scenario which is captured by the DIAG unit and recovered by the resource manager of the proposed approach. At the end, we also demonstrate a regular monitoring procedure in which the ports’ status are collected and sent to the resource manager by the MON unit.

A. Description of the Example

As shown in Figure 7, the platform is implemented as an SoC containing four tiles. In this scenario, the resource manager has been implemented by software on Tile1 and thus, TTEL1 acts as master and the other three TTELs act as slave, in the context of the resource management services. For the reconfiguration of each TTEL, a RECP port (shown as C in Figure 7) has been assigned (one local for TTEL1 and three remotes). Likewise, for the monitoring and diagnosis, MONP and ERRP has been used respectively (for each, one local and three remotes). The remaining tiles interact with each other via the dedicated communication channels (i.e., two RC channels from TTEL0 to TTEL2 and TTEL3 and two TT channels in the opposite side). MicroBlaze1 acts as a redundant for MicroBlaze0 and will be started, once a fault is detected in the main tile.

B. Fault Scenario

Figure 8 depicts a fault scenario, in which the driver running on MicroBlaze0 tries to write on the TT port (enq = 1), while the port is full (port_full = 1). As described in Section IV, the persistence of the driver to write on a full port is identified as a fault and changes the port_full_err signal from 0 to 1 and consequently, the respective bit in error_flags. The RMI identifies the faulty port by checking the flags and asks the port to send its data, by declaring the port_id on the respective bus (in our example, ID = 1). Once the diagnosis interface inside the port detects its ID on the bus, it puts the error_data which consists the error code, port address. Thereafter, the RMI sends this message to the ERRP (by asserting the write and putting the data on data bus) and terminates this process by asserting the terminate signal after 11 clock cycles.

C. Fault Recovery

Figure 9 illustrates the process of the fault-recovery. As the application running on MicroBlaze0 has been detected as a faulty subsystem, the ERRP at TTEL2 sends an error message to TTEL1 and informs the resource manager about an overflow error at port1. Based on the implemented algorithm, the resource manager decides to disable the port to prevent MicroBlaze0 from sending more messages. The process of disabling the faulty port starts by sending a message to the respective RECP which is connected to the RMI of TTEL3 through the defined communication channels. Once the message arrives at TTEL3, the RMI will be notified (new_conf = 1). Then, the RMI fetches data, checks validity and decodes them. By decoding the data, the RMI finds the port which should be disabled and finally after 14 clocks, the given port is disabled. As shown in the figure, after disabling the faulty port, the driver running on the core tries to write on the port (enq = 1) but the internal sig_enq is not affected, as the port is deactivated.

D. Monitoring Scenario

The process of collecting and sending the port status is performed periodically (triggered periodically by the freq signal in Figure 10). The RMI asks individual ports to put their status on the shared port_data bus, by declaring the ID of the port (on port_id bus). At the same time, the RMI writes the received data on the MONP and terminates the data by time-stamp (cf. write, address and data).
VII. CONCLUSION-DISCUSSION

In this work, we addressed the challenge of fault-recovery (addressed in safety-critical applications) in conjunction with adaptive QoS (which are widely used in low-critical applications) on a single shared hardware platform, where applications of difference criticalities coexist and interact. We addressed the monitoring and reconfiguration services at the chip-level communication, to enable the local resource manager to detect the fault and to dynamically reconfigure a resource, based on the monitored artifacts at the resource.

This section discusses how the presented solution meets the requirements of a reconfigurable architecture, that was addressed in Section VI.

a) Safe Reconfiguration: Since the monitoring and reconfiguration are performed through the ports [11], the whole procedure is encapsulated and temporally segregated, thus no interference could happen.

b) Bounded Time for Reconfiguration: As the reconfiguration is performed by hardware, the needed time for this operation is bounded and the shortest possible time (cf. Section VI, totally 24 clock cycles for detecting the fault and blocking the faulty core).

c) Consistent Switch to new Configuration: The reconfiguration command is sent to the LRSs through a TT message, whose jitter is minimal and the possibility of inconsistent reconfiguration is avoided by the underlying hardware.

d) Continuity of Service after the Reconfiguration: This requirement is satisfied by restricting the reconfiguration at each resource. For instance, the received values for the MINT service are encapsulated, as they are built on top of communication services, thus the reconfiguration mechanism is robust.

e) Robust Reconfiguration Mechanisms: The communication between the elements of monitoring and reconfiguration services are encapsulated, as they are built on top of communication services, thus the reconfiguration mechanism is robust.

f) State Preservation: Switching to a new configuration is performed, while there is no ongoing communication through the channel to avoid corruption of the messages. This is technically achieved by the definition of a global idle state, during which no communication is allowed.

ACKNOWLEDGMENT

This work has been supported by the European project DREAMS under the grant agreement no. 610640.

REFERENCES