GeMRTOS: Multiprocessor RTOS

User and Development Manual — Chapter 3: GeMRTOS Architecture

gemrtos.com

August 2026

Revision: 0.9a-2026-06-01

CHAPTER 3

GeMRTOS Architecture

3.1 System Overview

GeMRTOS is designed to facilitate real-time multitasking on multiprocessor systems. A key aspect is the management of hardware resources and the software architecture. Figure 3.1 shows a layout of the GeMRTOS architecture that compromises hardware and software. This innovative architecture adapts to evolving system requirements, ensuring that complex, real-time applications can be created through simplified processor management, synchronization, and resource allocation. Leveraging an event-driven design and hybrid partition scheduling, GeMRTOS empowers developers to build responsive and scalable applications for demanding environments, all while maintaining a clear separation between hardware and software components.

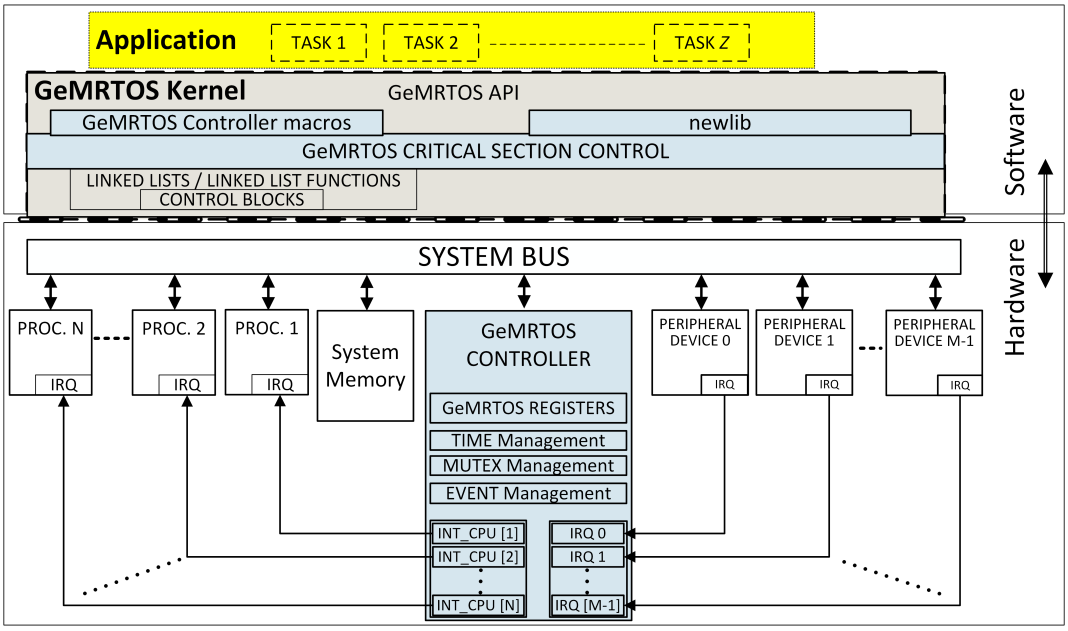

Figure 3.1: GeMRTOS architecture layout.

The hardware architecture of a GeMRTOS-based system includes the following core interconnected elements:

- **Processors:** Multiple processors (PROC 1, PROC 2, ..., PROC N) with unique `CPU_IDs`. These processors are the primary execution units where tasks run. Each processor has a unique CPUID (Central Processing

Unit Identifier) for identification on the bus and for use within the GeMRTOS itself. Current GeMRTOS version supports both the Nios II and the Nios V soft-processors for the Altera Field-Programmable Gate Array (FPGA) families.

- **System Bus:** This component is essential to allow communication between the multiple elements. Altera Nios II and Nios V processors utilize the Avalon bus.
- **Central Memory:** Accessible through the System Bus. This is the main memory used by all processors for code, data, and stacks. The size is dictated by the embedded application and specific hardware implementation.
- **GeMRTOS Controller:** Manages all the processor synchronization, time management, and Centralized Interrupt Management.
- **I/O Devices with Interrupt Requests (IRQs):** Connected to the Interrupt lines of the GeMRTOS controller. This provides a centralized, scalable and managed way to handle peripherals.

The software architecture of GeMRTOS relies on Control Blocks, linked to maintain system state, that store information about tasks, events, signals, resources, times, and processors. To ensure consistency, only functions holding the GeMRTOS Mutex in the GeMRTOS controller can modify these Control Blocks and their relationships. User applications are built from tasks, implemented as either periodically released functions or infinite-loops that suspend themselves to achieve regularly timed execution.

GeMRTOS maintains the system state using its controller registers and highly linked lists of control blocks. System state transitions occur in response to events. The GeMRTOS controller manages external events originating from peripherals via interrupts and maintains system time for generating timed events. The GeMRTOS controller then interrupts the most appropriate processor to service the event, updating system state accordingly. For instance, when a timed event occurs, the processor executing the task associated with the timed event, if any, is interrupted; otherwise, if the associated task is not executing, the processor executing the lowest priority task is interrupted.

In GeMRTOS, system processors are dedicated to executing application tasks and updating system state in response to events. System functions are executed only when system state needs to be updated, avoiding unnecessary periodic execution. The GeMRTOS controller and Control Blocks are essential for preserving system state and managing system events, a design that makes GeMRTOS efficient and flexible for real-time applications.

Before delving into the implementation details, it's essential to grasp the core concepts upon which GeMRTOS is built. This section provides an overview of these foundational elements.

3.2 GeMRTOS Controller

The GeMRTOS controller's primary function is to provide innovative and adaptable management of time and external events, as well as ensuring mutual exclusion among system processors to guarantee coherent access to the system state. This is achieved through a combination of hardware and software mechanisms, allowing for efficient and reliable coordination in a multiprocessor environment. The controller plays a critical role in maintaining system-wide consistency and enabling predictable real-time behavior.

3.2.1 Mutual Exclusion Among Processors

To preserve system state consistency, the GeMRTOS controller employs a hardware Mutex resource, an innovative approach that surpasses traditional locking mechanisms in efficiency. As a critical section, the system state requires that only one procedure be executed at a time, preventing conflicting modifications. Consequently, only one processor is permitted to modify the system state at any given time. The GeMRTOS controller implements this hardware Mutex resource, allowing only one processor at a time to execute system update procedures.

3.2.2 Time Management

Time management is crucial in real-time systems, dictating when events occur. The GeMRTOS controller leverages two primary registers to manage time with exceptional precision and adaptability. First, the System Time register increments based on the system clock and Prescale register configuration, typically set to count in 10MHz increments. This register tracks time in non-frozen mode. Second, the Frozen Mode Time register increments when the system switches to frozen mode, which is activated when the frozen mode is enabled and the next timed event exceeds the System Time by a threshold predefined by the user.

3.2.3 Interrupt Management

GeMRTOS employs an adaptable interrupt management architecture that allows the system to respond effectively to external events. In most cases, an application is implemented as a set of tasks executed by system processors, triggered by external events. The system includes both processors and peripherals; the processors interact with the external environment via these peripherals to receive stimulus. Interrupts are usually implemented to force a processor to execute the task that services the peripheral. GeMRTOS handles interrupt requests classified as:

- **Timed event interrupts:** These events are configured to happen at a certain system time. Time events are sorted according to the occurrence time in a total order relationship. The next time event is the earliest time event.
- **External interrupts:** These events are produced by the input/output peripheral devices. Most of these events require the execution of the associated interrupt service routine (ISR) in order to satisfy the demand of the input/output peripheral. Interrupt requests may be enabled or disabled by setting or clearing, respectively, each interrupt request in the GeMRTOS controller. Therefore, an interrupt request from the peripherals will produce a processor interrupt request when the interrupt request is enabled.
- **Internal interrupts:** A processor may interrupt another processor. However, this method should be limited only to internal GeMRTOS kernel procedures.

This innovative approach enables the GeMRTOS controller to effectively manage interrupt requests from timed events, system peripherals, and interval interrupts. By selecting the most adequate processor to execute the service task(s) associated with the interrupt, GeMRTOS optimizes system performance and responsiveness.

Furthermore, GeMRTOS implements triggers, a sophisticated mechanism that expands upon traditional interrupt handling. Triggers are associated with both internal and external events, releasing one or several tasks in response. This adaptable design allows an event to trigger the execution of multiple associated tasks, while also supporting timeout settings to handle events that do not occur within a predefined time interval.

3.3 System Components and Control Blocks in GeMRTOS

GeMRTOS achieves its real-time capabilities and adaptable behavior through a set of carefully designed system components and control blocks. In contrast with traditional methods, GeMRTOS generalizes all the functions and system components into a set of structures whose information is hold into control block highly linked among them. GeMRTOS relies on control blocks to maintain the system's state. Key control blocks include:

- **Kernel Control Block** (`KCB`): Centralizes operating system information.
- **Task Control Block** (`TCB`): Stores runtime information for an application task.
- **Event Control Block** (`ECB`): Manages event-related information.
- **Scheduling List Control Block** (`LCB`): Supports hybrid partition scheduling.
- **Processor Control Block** (`PCB`): Tracks the status of each system processor.
- **Resource Control Block** (`RCB`): Manages system resources like semaphores and message queues.
- **Signal Control Block** (`SCB`): Defines internal signals.

This innovative design, comprised of carefully linked control blocks, maintains the system's state throughout runtime. Only the processor that currently holds the GeMRTOS Mutex is authorized to modify the information held within these control blocks, ensuring system consistency and data integrity. Each control block is associated with a key entity within GeMRTOS, contributing to the system's modularity and adaptability.

3.3.1 Task

In GeMRTOS, a task is defined as a fundamental unit of execution, designed for flexible and adaptable management within a multiprocessor environment. As illustrated in Figure 3.2, each task consists of the following elements:

- **Task Control Block (TCB):** A control block storing essential runtime information for task management, enabling dynamic scheduling and resource allocation.
- **Task Code:** The executable code loaded into system memory, implementing the task's function when executed by a system processor.
- **Task Stack:** Data storage for preserving the task's status when suspended, enabling seamless resumption and efficient resource management.

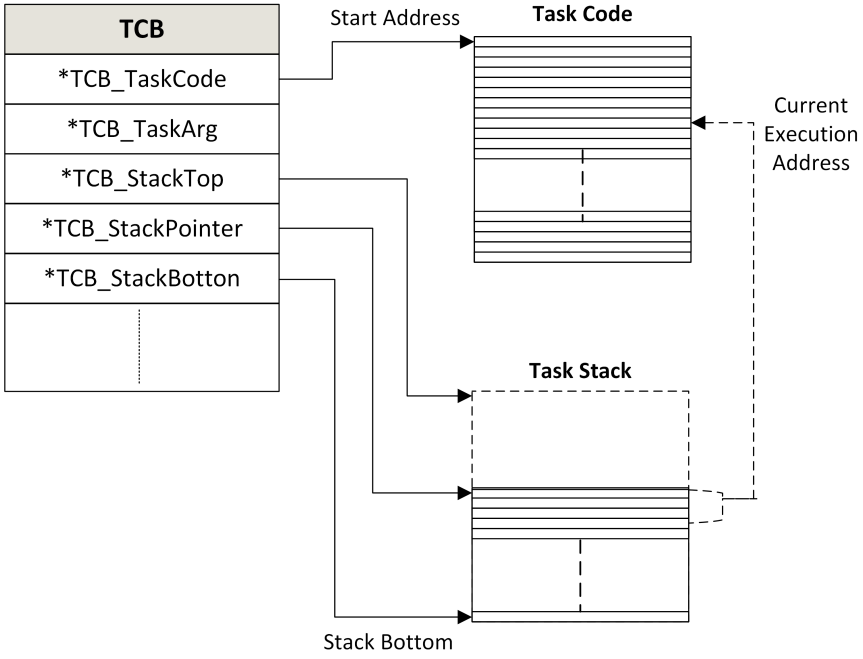

Figure 3.2: GeMRTOS task main components.

GeMRTOS tasks can exist in different states, each reflecting a particular stage in their lifecycle:

- **Waiting State:** The task is awaiting the release of an event, enabling efficient synchronization and event-driven behavior. A task in this state is referred to as a "waiting task."
- **Ready State:** The task is prepared for execution, awaiting a processor to grant access. A task in this state is known as a "ready task," enabling adaptable scheduling based on system demands.
- **Running State:** The task is actively executing on a processor. Task can be executed to different states: suspended, finishes the execution and it is preempted by a higher priority task. A task in this state is referred to as an "executing task."

Tasks transitions states according to the events that change states. To support adaptable system behavior, GeMRTOS defines a set of event categories that trigger task state changes:

- **Create Events:** Create a task and set its state to waiting, initializing its stack for execution.

- **Release Events:** Change the task state from waiting to ready, enabling processor allocation based on task priority.
- **Suspend Events:** Change the task state from ready to waiting, preserving its context for later resumption.
- **Blocking Events:** Change the task state from running to waiting, typically due to resource contention or synchronization requirements.
- **Execution Events:** Change the task state from ready to running, allowing a processor to execute the task code.
- **Deallocate Events:** Change the state of a task from the running state to the ready state.
- **Finish Events:** Destroy a task. A finish event destroys a task-independent of its state.

Tasks may be classified as:

- **User Task:** Tasks released for execution according to user-defined configurations.
- **Idle Task:** Tasks executed when no other user tasks are ready.

This innovative task management framework, leveraging flexible state transitions and event-driven mechanisms, enables GeMRTOS to efficiently orchestrate complex applications within a multiprocessor environment. By dynamically adapting to changing system conditions and prioritizing critical tasks, GeMRTOS ensures responsive and reliable performance in demanding real-time scenarios.

3.3.2 Events

GeMRTOS leverages an event-driven architecture, enabling adaptable and efficient responses to both internal and external stimuli. Events are the cornerstone of system activity, triggering task state transitions and coordinating resource allocation. By responding to events rather than relying on constant polling, GeMRTOS minimizes overhead and maximizes system responsiveness. GeMRTOS supports three primary categories of events:

- **Timed Events:** Occur when the system time reaches a specific, pre-defined value. These events enable precise scheduling and control of time-critical operations, offering a predictable and reliable mechanism for tasks that must execute at specific intervals.
- **External Events:** Generated by peripheral devices, signaling asynchronous interactions with the external environment. These events enable the system to react promptly to changes in its surroundings, facilitating responsive control and monitoring of external processes.
- **Internal Events:** Occur based on runtime conditions within the system, including synchronizations and signals. These events allow for adaptable responses to unforeseen circumstances and dynamic system adjustments, ensuring robust and reliable operation even under unexpected conditions.

Through this comprehensive event management framework, GeMRTOS provides a highly adaptable and responsive platform for building complex, real-time embedded systems. By carefully defining and managing event triggers, developers can create systems that react efficiently to changing conditions, optimize resource utilization, and maintain reliable performance in demanding environments.

3.3.3 Time

Time is a fundamental element in GeMRTOS, providing the crucial foundation for time-aware decision making and efficient task scheduling. In an innovative approach to time management, the GeMRTOS controller utilizes two key registers: the `R_TM_CNT` register, which holds the system time in non-frozen mode, and the `R_FRZ_TM_CNT` register, which holds the length of time the system stayed in frozen mode.

This dual-register system enables adaptable time management in the face of varying system loads. The GeMRTOS controller intelligently produces a time event each time the `R_TM_CNT` register reaches the value stored in the `R_NXT_EVN_CNT` register. Upon reaching this value, GeMRTOS interrupts either the processor executing the task associated with the timed event (if it is executing) or the processor executing the lowest priority task (if no task

is associated with it). This adaptable approach ensures that timed events are efficiently processed by choosing the most suitable processor to interrupt, balancing responsiveness with overall system load.

Through this innovative time management system, GeMRTOS provides a robust and adaptable foundation for building real-time applications. By precisely tracking system time and intelligently scheduling time events, GeMRTOS empowers developers to create systems that meet strict timing deadlines and maintain reliable performance under dynamic conditions.

3.3.4 Resources

In a multitasking system, tasks often share access to system resources, such as data memory, code memory, I/O devices, and specialized hardware components. To manage the utilization of these resources efficiently and fairly among tasks, GeMRTOS provides an innovative resource management mechanism that enables adaptable resource sharing and prevents resource contention. This mechanism offers data structures and functions designed to handle the sharing of system resources among the tasks that require their utilization.

GeMRTOS implements the Resource Control Block (RCB) to manage the events associated with resources, supporting a variety of resource management strategies. The Resource Control Block provides a flexible approach to assigning resources. In fact, when a task requests a resource, one of two approaches can be applied (1) the resource may be granted to the task if it is available or (2) the task may be blocked if it must wait until the resource becomes available. Moreover, a timeout may need to be specified in order to avoid a task either granting or waiting for a resource for a long time, which would impede the adequate sharing of resources among tasks.

3.3.5 Signals

In operating systems, signals provide a powerful mechanism for handling exceptional conditions that may arise during runtime, offering an adaptable and efficient way to respond to unforeseen events. GeMRTOS leverages signals to enable tasks to execute associated code when a specific condition occurs, facilitating flexible error handling and dynamic system adjustments. These conditions may be predefined, such as task creation, resource allocation, or task blocking. The Signal Control Block (SCB) control structure is defined to store signal information, and is linked to control block structures accordingly. The SCB control structure points to the code to be executed when the condition of the signal occurs. This code is executed prior to the execution of the code of the task with which the signal is associated. This innovative approach enhances system robustness and adaptability by providing a structured way to handle exceptional circumstances.

3.3.6 Processors

The system processors are the workhorses of GeMRTOS, executing the code of the operating system, tasks, and signals. The operating system is in charge of sharing the utilization of the processors among the system tasks. This adaptable processor allocation is achieved through an innovative approach that ensures efficient resource utilization and responsiveness. Each processor is assigned to execute the highest priority tasks in the ready state. The Processor Control Block (PCB) stores the state of each processor, enabling the system to dynamically track processor availability and workload. Each processor may serve several scheduling lists, and several processors may serve the same scheduling list, allowing for a highly flexible and scalable task distribution.

3.3.7 Hybrid Scheduling Lists

GeMRTOS introduces an innovative hybrid scheduling approach, offering exceptional configuration flexibility for diverse application requirements. Unlike traditional scheduling methods, GeMRTOS allows application

configuration using hybrid scheduling lists, enabling dynamic adaptation to changing system conditions. Each task is assigned to a hybrid scheduling list but may migrate among lists during runtime, offering unprecedented control over task placement. Processors are assigned to one or more hybrid scheduling lists, allowing each scheduling list to be served by multiple processors, and each processor to serve multiple lists, creating a highly flexible and adaptable scheduling environment. This configuration enables efficient and sophisticated load balancing strategies to be easily implemented. To avoid overload, each hybrid scheduling list allows defining an exclusion parameter that bound the number of processors that may simultaneously execute tasks assigned to that hybrid scheduling list. The List Control Block (LCB) structures hold all the information relative to each hybrid scheduling list, and several hybrid scheduling lists may be created. Fixed-priority and Earliest-deadline-first priority disciplines may be defined for each hybrid scheduling.

3.4 GeMRTOS Code Structure

GeMRTOS code structure defines how tasks are executed during runtime, ensuring a synchronized and adaptable system operation. All critical sections (shadow containers) of GeMRTOS are executed by granting the GeMRTOS Mutex, maintaining consistency when system status is updated. This innovative approach provides a predictable and reliable framework for managing task execution and preserving data integrity. As depicted in Figure 3.3, the code structure comprises several key sections.

The main sections of the code structure include:

- **Initialization sections:** During initialization, the GeMRTOS controller manages processor startup, keeping all but the processor with CPUID 1 in reset. This processor initializes core data structures, interrupt handlers, and executes the user-defined `main()` function. The remaining processors are then sequentially released from reset after being granted the GeMRTOS Mutex, guaranteeing an orderly initialization process.
- **Task start section:** After acquiring the GeMRTOS Mutex, the Task Start Section initializes task execution on a processor by clearing its status, selecting the highest-priority ready task or the idle task, updating the PCB, setting the processor priority in the GeMRTOS controller, handling priority inversions with trigger interrupts, updating the processor of the next time event and calling the `START_TASK` macro to begin execution.
- **Interrupt handler section:** When a processor receives an interrupt request from the GeMRTOS controller, it saves its state, disables the PIRQ, acquires the GeMRTOS Mutex, and then reads the event register to determine the appropriate action: execute the `gk_UnFrozenCallback`, the `gk_FrozenCallback` or `gk_TimeCallback` functions, or disable a DIRQ and release its trigger, and call the start section to proceed with the next task.
- **System call section:** To ensure consistent operation, system calls invoked by tasks execute within the GeMRTOS controller's critical section, releasing the GeMRTOS Mutex if no context switch is needed, otherwise invoking the start task section and guaranteeing completion upon task resumption.
- **Tasks codes:** Are executed by system processors outside the critical section of the GeMRTOS controller, but the critical section may be requested when a task code calls a system call. The task stack stores the data and information required for the task code to be executed during runtime. A TCB control block is associated with each task to hold all the task status required by the task for execution.
- **Task suspend section:** Upon task suspension, the system removes the Task Control Block (TCB) from the executing list and adds it to the waiting list, selects a new ready task for execution by the current processor, and then calls the start task section to initiate the new task.
- **Task complete section:** When a task completes, its TCB is moved to the waiting list, and if it was an Interrupt task, its DIRQ is re-enabled.

This carefully orchestrated code structure, encompassing distinct sections for initialization, task management,

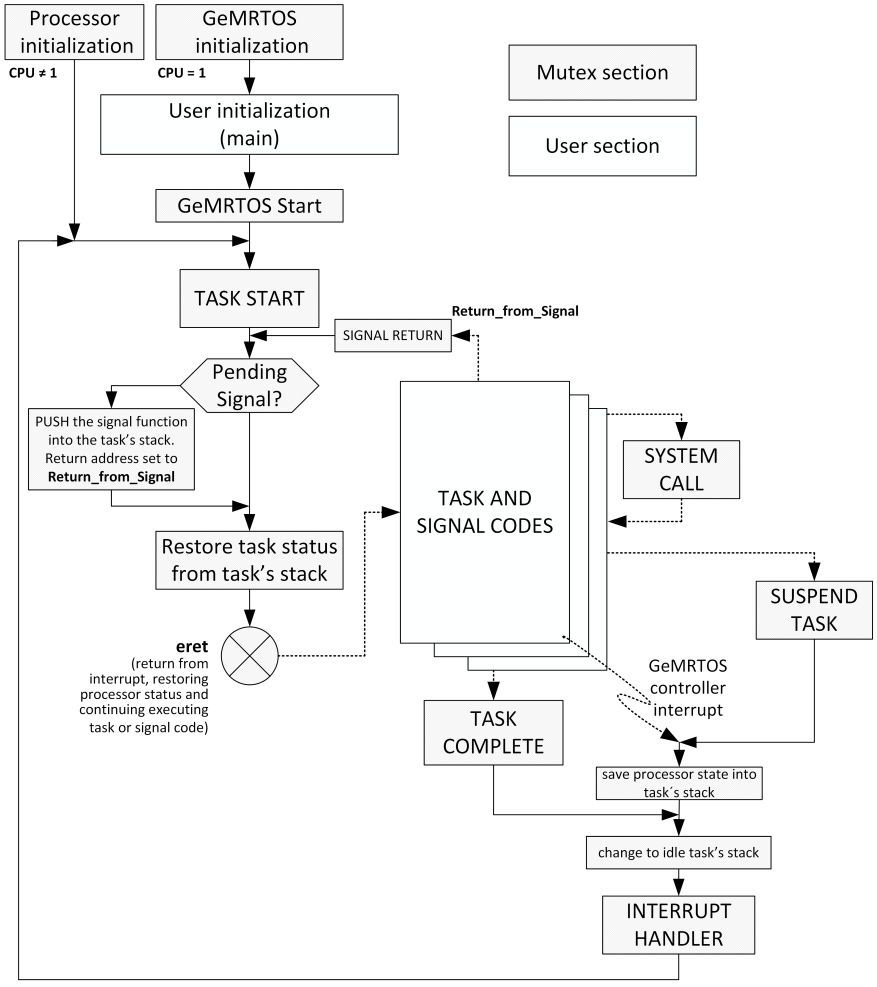

Figure 3.3: GeMRTOS code structure.

interrupt handling, and system calls, provides GeMRTOS with a robust and adaptable framework for executing real-time applications on multiprocessor systems. The subsequent sections detail each of these components and the flow of each of them.

3.4.1 Initialization Sections

During initialization, GeMRTOS creates and configures its core Control Blocks, establishing the foundation for adaptable and efficient system operation. The GeMRTOS controller manages processor startup, innovatively keeping all processors except the one with CPUID 1 in a reset state. After each processor initializes successfully, the GeMRTOS controller releases the next processor in sequence, ensuring a synchronized and orderly startup process. This approach leverages a controlled release mechanism to guarantee consistent initialization across all processors, a key innovation in GeMRTOS's multiprocessor design.

The processor with CPUID 1 is responsible for system-wide initialization:

- It initializes the GeMRTOS controller's internal registers, including the system time.
- It creates and configures all core Control Blocks.
- It sets up the initial state of all linked lists.
- It creates interrupt handlers by associating a trigger resource with each peripheral interrupt request (DIRQ), and an idle task for each processor.
- Finally, it executes the user-defined `main()` function, allowing the application to initialize its own tasks and resources.

To ensure synchronized startup, only the processor with CPUID 1 performs this configuration. All other processors remain in reset until they are granted the GeMRTOS Mutex by the previously initialized processor, guaranteeing orderly initialization before application tasks begin execution.

3.4.2 Task Start Section

The Task Start Section, executed after acquiring the GeMRTOS Mutex, plays a crucial role in initiating task execution on a processor with an innovative approach to managing the resources. In this section, the following steps are executed:

- The processor's status in its Processor Control Block (PCB) is cleared, indicating no assigned task.
- The processor searches its assigned hybrid scheduling lists in priority order to find the highest-priority task in the ready state. If no ready task is found, the processor's idle task is selected.
- The processor's PCB is updated with the selected task as the currently executing task. The task's Task Control Block (TCB) is linked to the running list of its hybrid scheduling list, based on the task's running priority.
- To support preemption, the processor priority equal to the running task priority is then updated in the GeMRTOS controller, for handling external or internal events.
- Priority inversions are checked and handled by enabling trigger interrupts on processors running lower-priority tasks blocking higher priority one with the same LCB control block.
- The processor that will execute the next timed event in the register of the GeMRTOS controller is updated, which processor associated with the TCB control block linked to the first ECB control block of the GeMRTOS time list.

This innovative approach facilitates dynamic adaptation to changing system demands. By selecting the highest-priority ready task and configuring the processor accordingly, GeMRTOS ensures efficient resource utilization and predictable real-time performance, all while maintaining a clear and structured task execution framework.

3.4.3 Interrupt Handler Section

The Interrupt Handler Section contains the code executed when a processor receives a Processor Interrupt Request (PIRQ) from the GeMRTOS controller, resulting in a processor interrupt signal. This innovative section ensures timely response to system events while maintaining system integrity. In this section, the processor performs the following steps: saves the processor state into the current executing task stack, disables its interrupts, request the GeMRTOS Mutex and reads then Event register of the GeMRTOS controller to get the event code. Depending on the event code, the following occurs:

- `EVN_CODE_FROZEN`: The `gk_FrozenCallback` function is called to handle the switching of the system to the frozen mode.
- `EVN_CODE_UNFROZEN`: The `gk_UnFrozenCallback` function is called to handle the switching of the system to the non-frozen mode.
- `EVN_CODE_TIMED`: The `gk_TimeCallback` function is called to process the interrupt associated with the timed event.
- `DIRQI_CODE`: The requesting peripheral interrupt request (`DIRQ`) is disabled in the GeMRTOS controller, and the associated trigger is released.

This structure allows GeMRTOS can support processors lacking native interrupt functionality. The processor checks for interrupt requests in the GeMRTOS controller when tasks are suspended or completed. To reduce interrupt latency, any processor can be assigned to wait state mode for such events, reducing the latency to near zero.

In summary, the interrupt handler section ensures that the processor safely responds to system events by saving its state, processing the appropriate routine based on the interrupt cause, disabling further interrupts, and relinquishing control to the next scheduled task, ensuring system stability and responsiveness.

3.4.4 System Call Section

GeMRTOS ensures the integrity and consistency of system operations through its unique approach to handling system calls. Instead of letting tasks directly modify sensitive system components, GeMRTOS requires that all system calls execute within the protected "critical section" of the GeMRTOS controller. Imagine it as a specialized processing zone where tasks must temporarily venture to request services from the system core. This innovative design choice guarantees that all modifications to GeMRTOS data structures and resources are performed in a tightly controlled and synchronized manner, shielding the system from data corruption and maintaining overall stability.

The journey a system call takes depends on its specific purpose. Some calls are simple and direct, requiring no change in context. In these cases, the GeMRTOS Mutex is released before returning control to the calling task, minimizing overhead. However, when a system call requires a more complex operation, such as switching to a different task, the "start task section" is invoked. This innovative process ensures that when the original task resumes, the system call will seamlessly complete its execution, preserving the integrity of the overall operation. By mandating that all state-modifying actions happen via this protected path, GeMRTOS ensures that resources, macros, and system primitives are managed reliably and predictably.

3.4.5 Task Suspend Section

The Task Suspend Section defines the adaptable process by which GeMRTOS manages task suspension, ensuring responsiveness and maximizing processor utilization. When a task's execution must be temporarily paused, GeMRTOS initiates a controlled transition. The Task Control Block (TCB) of the currently executing task

is removed from the executing list and added to the waiting list. This action effectively places the task in waiting state, awaiting a future event to trigger its resumption.

To maintain system momentum, the current processor selects a new task from the ready tasks eligible for that processor. This ensures processors always have tasks to serve and allows a seamless transition to the start task section to initiate the new task.

3.4.6 Task Complete Section

The Task Complete Section outlines the systematic process executed when a task finishes its execution, guaranteeing resource cleanup and enabling a streamlined transition to subsequent tasks. The point in which this section is reached at task completion is when a task finishes and is removed. The following key steps are then executed:

- The task's TCB is removed from the executing list, allowing the processor to be reallocated.
- The task's TCB is linked to the waiting list, preserving the task's state for potential future use.
- If the task was an Interrupt task, the corresponding peripheral interrupt request (DIRQ) is re-enabled to allow for further interrupts.
- The scheduler selects the highest priority ready task for the current processor, ensuring continuous processor utilization.
- The start task section is invoked to execute the selected task, initiating the next task's execution cycle.

This structured process ensures that upon task completion, the system removes the task from the execution list, returns it to the waiting list, re-enables the DIRQ for Interrupt tasks, and then initiates the start task routine to select and execute the next ready task. This contributes to the GeMRTOS efficient use of the resources.

3.5 GeMRTOS Linked Lists

GeMRTOS leverages a sophisticated system of linked lists to maintain system information and preserve system consistency during runtime. This approach is a major component since they enable a highly adaptable, event-driven architecture, allowing the system to respond efficiently to changing conditions. Several linked list structures are established among control block structures, forming the backbone of GeMRTOS's dynamic operation. The control block linked lists are basic in GeMRTOS and crucial in all applications. These linked lists helps to preserve the status of the system.

The following subsections provide a brief overview of several key linked list structures, illustrating how GeMRTOS maintains system status and enables adaptable operation.

3.5.1 Hybrid Scheduling List Structure

GeMRTOS offers a very flexible scheduling mechanism based on hybrid scheduling lists. This innovative approach allows for dynamic task allocation and efficient processor utilization, adapting to changing system conditions and application requirements. Figure 3.4 shows an overview of the main links involved in the hybrid scheduling list structure.

Key components of this structure include:

- **LCB Control Block:** The base of the hybrid scheduling list architecture, holding information about the hybrid scheduling list. Each hybrid scheduling list will have a LCB assigned to it in order to hold all the information about the hybrid scheduling list. The root of this linked list is the `KCB_NextLCBL` of the

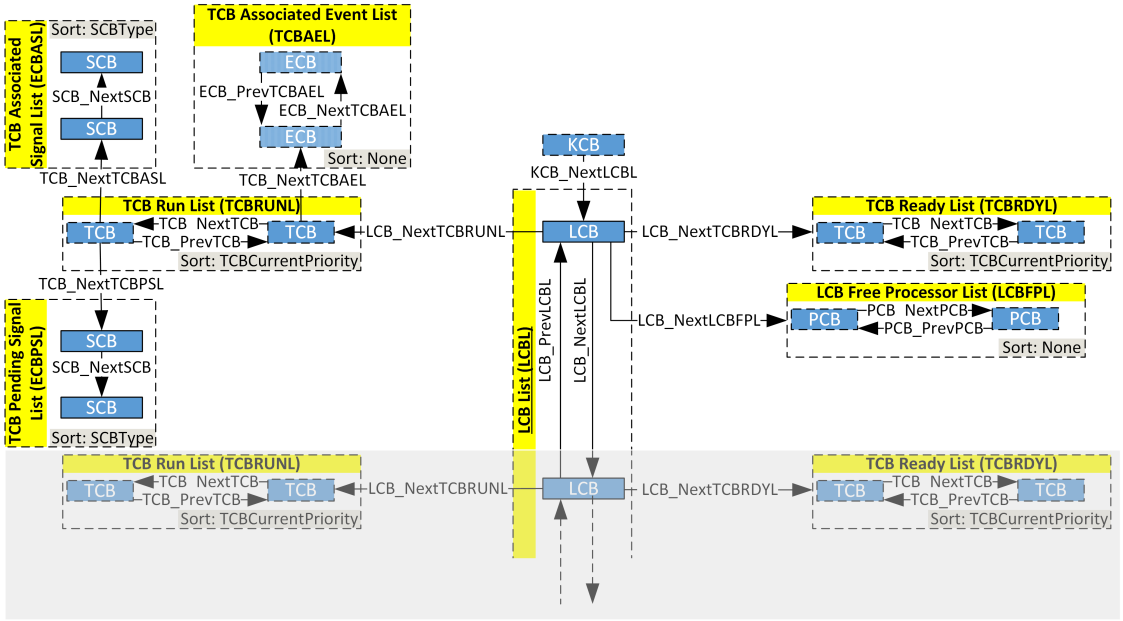

Figure 3.4: Hybrid scheduling list overview.

KCB structure, and the links between the LCB control blocks are defined by the `LCB_NextLCBL` and `LCB_PrevLCBL` fields of each LCB control block.

- **TCB Run List:** Links all TCBs of tasks in the running state assigned to the hybrid scheduling link. The root of the linked list is the `LCB_NextTCBRUNL` field of the LCB control block, and the TCB are dual linked with the fields `TCB_NextTCB` and `TCB_PrevTCB` of each TCB sorted in ascendent order by the field `TCBCurrentPriority`.
- **TCB Ready List:** Links all TCBs in the ready state assigned to the hybrid scheduling list. The root of the linked list is the `LCB_NextTCBRDYL` field of the LCB control block and the TCB are dual linked with the fields `TCB_NextTCB` and `TCB_PrevTCB` of each TCB sorted in ascendent order by the field `TCBReadyPriority`.
- **LCB Free Processor List:** The field of the LCB control blocks points to the LCB Free Processor list that links all the free processor assigned to the hybrid scheduling list.

Each TCB control block of tasks either in ready or running state may have links with other control blocks that may include:

- TCB Associated Event List: a list of ECB control blocks that may linked to resource associated lists or the time event list.
- TCB Associated Signal list: a list with all the SCB control blocks of the signals created for the task.
- TCB Pending Signal list: a list with all the TCB of the signals that are pending to be executed the next time the task goes to running state.

In summary, the hybrid scheduling partition list structure provides a flexible and adaptable framework for managing tasks and processors in GeMRTOS. By allowing dynamic task migration, efficient load balancing, and adaptable processor assignments, this innovative approach enables developers to create high-performance real-time applications that can respond effectively to changing system demands.

3.5.2 Resource Management Lists

GeMRTOS implements a generic structure to manage all the resources. A resource management list is based on the RCB control block. Figure 3.5 shows an overview of the main linked list involved in the resource management

of GemRTOS.

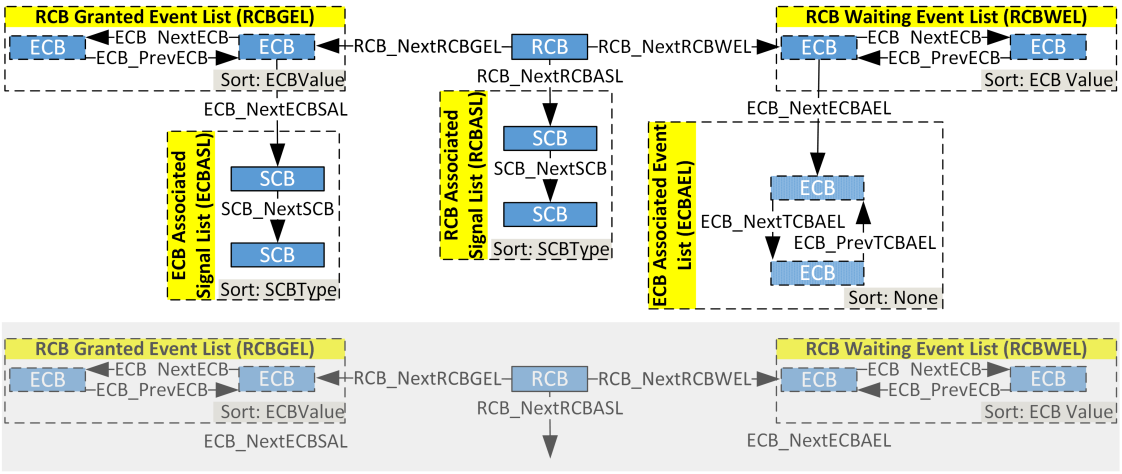

Figure 3.5: Resource management lists.

Key lists involved in the resource management include:

- **RCB Waiting Event List (RCBWEL):** This list links all the events associated with tasks waiting for a specific resource (e.g., a mutex). For a mutex resource, the RCBWEL links all events associated with tasks waiting for the mutex.
- **RCB Granted Event List (RCBGEL):** This list links all the events associated with tasks that have been granted a specific resource. For a mutex resource, the RCBGEL links all the events associated with task granting the mutex.

The linked Events in these lists are also linked with TCB to control a set of the associated tasks. In addition, ECB control blocks may also have signal control blocks linked to them in order to produce a signal in the tasks associated with those events. By using this architecture the GeMRTOS controller implement timeouts associated with events.

In summary, GeMRTOS's resource management lists provide an innovative and adaptable framework for managing access to shared resources. By dynamically linking tasks, events, and resources, GeMRTOS enables efficient resource utilization, prevents contention, and facilitates the development of robust and responsive real-time applications.

3.5.3 ECB Time List (ECBTL)

The ECB Time List (ECBTL) plays a vital role in GeMRTOS by managing timed events efficiently. Its flexible structure allows for precise scheduling and execution of time-critical operations. Figure 3.6 provides an overview of the main components of the ECBTL.

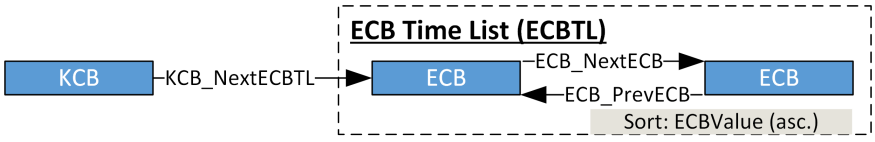

Figure 3.6: ECB Time List (ECBTL) structure.

The ECBTL incorporates several essential features:

- **Sorted Structure:** All timed Event Control Blocks (ECBs) are linked in the ECBTL, sorted by their occurrence times (stored in the `ECBValue` field of each ECB control block).
- **Efficient Access:** The first ECB in the linked list represents the closest timed event. Its time value is consistently copied to the `NEXT_OCCURRENCE_TIME` register of the GeMRTOS controller, ensuring rapid access and precise timing.
- **Centralized Management:** The `KBC_NextECBTL` field of the Kernel Control Block (KCB) serves as the root of the list, centralizing time event management within the kernel.

By organizing ECBs according to their scheduled execution times, GeMRTOS efficiently determines the next event requiring processing. This innovative approach, combined with the `NEXT_OCCURRENCE_TIME` register, enables precise and adaptable scheduling of time-critical operations. As a result, GeMRTOS ensures reliable real-time performance across various applications and scenarios.

3.6 Conclusions

GeMRTOS represents a cutting-edge, highly adaptable solution for real-time multitasking on multiprocessor systems. This chapter has delineated the core structures and fundamental principles that define its architecture. The foundation of GeMRTOS is built upon four key elements:

- **GeMRTOS Controller:** This component efficiently manages external events (IRQs), processor startup, mutual exclusion (Mutex) among processors, time events, frozen and unfrozen modes, and interrupt handling.
- **GeMRTOS Control Blocks:** These highly linked data structures maintain comprehensive information about all entities within GeMRTOS, preserving system state throughout runtime.
- **GeMRTOS Code Structure:** This framework synchronizes task execution across all system processors, ensuring consistency and predictable behavior.
- **GeMRTOS Linked Lists:** These structures facilitate adaptable system management in an event-driven manner, enhancing efficiency and reducing overhead.

These core elements work in concert to enable a generic implementation of diverse mechanisms and resources, empowering GeMRTOS to excel in demanding real-time applications. The innovative design facilitates the natural implementation of sophisticated features, such as frozen and non-frozen modes. As a result, overall system performance and processor efficiency are significantly improved.

In the GeMRTOS architecture, processors are primarily dedicated to executing tasks and updating system status when events occur. The GeMRTOS controller assumes responsibility for event and time management, further enhancing system resource utilization. This strategic division of labor ensures that GeMRTOS can meet the exacting demands of complex real-time systems while maintaining flexibility and scalability.

By leveraging these advanced features and architectural design principles, GeMRTOS positions itself as a powerful and versatile solution for a wide range of real-time multiprocessor applications.