GeMRTOS: Multiprocessor RTOS

User and Development Manual — Chapter 4: GeMRTOS Controller

gemrtos.com

August 2026

Revision: 0.9a-2026-06-01

© 2026 GeMRTOS. All rights reserved.

ALTERA, AGILEX, ARRIA, CYCLONE, MAX, STRATIX, MEGACORE, NIOS, QUARTUS and PLATFORM DESIGNER are trademarks of Altera Corporation.

All other words and logos identified as trademarks or service marks are the property of their respective holders.

CHAPTER 4

GeMRTOS Controller

4.1 Introduction to the GeMRTOS Controller

The GeMRTOS controller is a pivotal component of the GeMRTOS architecture, providing advanced management of time, external events, and mutual exclusion among system processors. Its design ensures coherent access to the system state, facilitating robust and predictable system operation. The controller incorporates various configuration and status registers, which are mapped into the processors' memory space.

These memory-mapped registers are organized into two distinct sections, as illustrated in Figure 4.1:

The two memory sections are:

- `gemrtos_global_region`: This shared memory region is accessible to all system processors. It begins at the memory address `GEMRTOS_DRIVER_BASE` and contains control and status registers common to all processors.
- `gemrtos_processor_region`: This shared memory region provides each processor with access to an individual shadow memory area. The registers mapped to this region are specific to each processor and may contain different values depending on the accessing processor. For example, the `CPU_ID` register returns a unique identifier assigned to the processor reading the register. This memory region starts at the address `GEMRTOS_DRIVER_PROCESSOR_BASE`.

A comprehensive understanding of these memory regions and their associated registers is essential for effectively utilizing the GeMRTOS controller's capabilities. The subsequent sections will explore each region in detail, elucidating the functions performed by the GeMRTOS Controller during system operation.

This introduction provides a foundation for understanding the GeMRTOS controller's role in managing system resources and facilitating efficient inter-processor communication. The innovative design of the controller, with its distinct memory regions, contributes significantly to the overall performance and flexibility of the GeMRTOS architecture.

4.2 GeMRTOS Global Region

The `gemrtos_global_region` contains all the control and status registers of the GeMRTOS Controller common to all the processors. These registers enable coordinated system-wide management and provide a centralized point for configuring key parameters. Table 4.1 shows the registers according to the address offset referred to the `GEMRTOS_DRIVER_BASE` memory address.

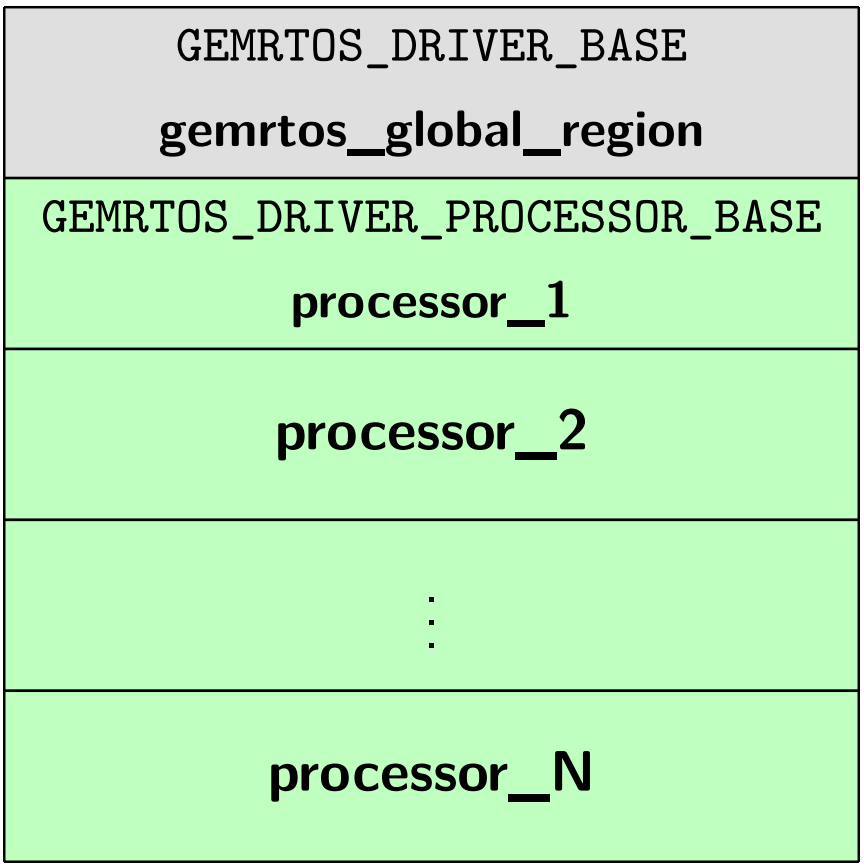

Figure 4.1: Memory-mapped regions of GeMRTOS Controller registers.

Table 4.1: Global region registers of the GeMRTOS controller.

Dword Offset	Name	R/W	Description
0x00-0x07	PRC_CDE_ADDRx	R/W	Reset Code for CPU_ID $\neq$ 1 PRC_CDE_ADDRx registers store the reset code for processors with CPU_ID different from 1.
0x08-0x0F	PRC_CDE_ADDRx	R/W	Interrupt Return Code The PRC_CDE_ADDRx registers also hold the return-from-interrupt instruction code and the sequence to restore the last set of registers.
0x10	STATUS_CTRL	R/W	Status Register (STATUS) The Control Register (CTRL) is accessed when reading. Control Register (CTRL) The Control Register (CTRL) is accessed when writing.
0x11	PRC_IRQ_ACK	R	PRC_IRQ_SECTION Register This register provides a mechanism for synchronizing interrupt handling across multiple processors in the GeMRTOS system. By checking the status of the PRC_IRQ_SECTION register, a processor can determine if the target processor has entered the IRQ section and is ready to process the interrupt.
0x12	IRQ_RQS_PND	R	IRQ_RQS_PND Register The IRQ_RQS_PND register returns the state of the interrupt requests from the system devices. This register provides a centralized view of all the pending interrupt requests in the GeMRTOS system.
0x13	Reserved	-	-
0x14	SYS_TM_HGH	R	SYS_TM_HGH Register The SYS_TM_HGH register holds the most significant word of the System Time register. This register, along with the SMP register, forms the complete 64-bit system time counter maintained by the GeMRTOS controller.
0x15	FRZ_THR	R/W	FRZ_THR Register The FRZ_THR register holds the most significant word of the frozen threshold value. This register, along with the SMP register, forms the complete 64-bit frozen threshold value used by the GeMRTOS controller.
0x16	FRZ_TM	R	FRZ_TM Register The FRZ_TM register holds the most significant word of the frozen time value maintained by the GeMRTOS controller. This register, along with the SMP register, represents the complete 64-bit frozen time counter. content...
0x17	TM_CNT	R	TM_CNT Register The TM_CNT register holds the most significant word of the non-frozen time counter maintained by the GeMRTOS controller. This register, along with the SMP register, represents the complete 64-bit non-frozen time value.

Continued on next page

Table 4.1: Global region registers of the GeMRTOS controller. (Continued)

Dword Offset	Name	R/W	Description
0x17	NXT_OCC_TM	W	NXT_OCC_TM Register The NXT_OCC_TM register is used to set the most significant word of the next occurrence time in the GeMRTOS controller. This register, in conjunction with the SMP register, allows the software to specify the complete 64-bit value for the next timed event.
0x18	ELAPSE_TM	R	ELAPSE_TM Register The ELAPSE_TM register is a 32-bit time counter that provides a mechanism for measuring elapsed time between two consecutive read operations. This register has a unique behavior that makes it particularly useful for performance monitoring and time interval measurements in the GeMRTOS system.
0x19	INOUTS	R/W	INOUTS Register The INOUTS register serves as a versatile interface for handling both inputs and outputs of the GeMRTOS controller. This dual-purpose register provides a convenient mechanism for the software to interact with external devices and signals, facilitating efficient I/O operations within the GeMRTOS environment.
0x1A	RST_CNT	R	RST_CNT Register The RESET register is a crucial component in the GeMRTOS controller that facilitates the coordinated and sequential start-up of multiple processors in a multi-core system. This register provides essential information for synchronizing the initialization process across different processor cores.
0x1B	SMP	R/W	SMP Register The SMP register is a crucial component in the GeMRTOS controller that facilitates 64-bit data transfers. This register works in conjunction with other registers to enable reading and writing of 64-bit values across the system.
0x1C	MTX_GRN	R	MTX_GRN Register The MTX_GRN (Mutex Grant) register is a crucial component in the GeMRTOS controller that manages mutex (mutual exclusion) allocation across multiple processors in a multi-core system. This register provides a bitmap representation of mutex ownership, allowing for efficient synchronization and resource sharing among processors.

4.2.1 Reset Code for CPU_ID $\neq$ 1

PRC_CDE_ADDRx registers store the reset code for processors with CPU_ID different from 1. This mechanism is implemented to overcome two common limitations:

- Restrictions in reset regions of certain processors
- Constraints in defining multiple reset vectors in compilation tools

By storing reset code for secondary processors in these registers, GeMRTOS ensures proper initialization across all cores in a multi-processor system, regardless of individual processor reset capabilities. This approach enables:

1. Flexible boot sequences for heterogeneous multi-core systems
2. Consistent initialization procedures across diverse hardware configurations
3. Enhanced portability of GeMRTOS to various processor architectures

The PRC_CDE_ADDRx registers thus play a crucial role in the robust and adaptable startup process of GeMRTOS in complex multiprocessor systems.

4.2.2 Interrupt Return Code

The PRC_CDE_ADDRx registers also hold the return-from-interrupt instruction code and the sequence to restore the last set of registers. This mechanism serves two important purposes:

1. It enables the processor to execute the interrupt return operation.
2. It ensures the release of the GeMRTOS Controller Mutex at the same time.

When a processor needs to return from an interrupt, it executes the stored return-from-interrupt instruction. This not only completes the interrupt handling, but also automatically releases the GeMRTOS Controller Mutex that was acquired during the interrupt processing.

This approach provides a tightly integrated and efficient way to handle interrupt returns. By coupling the interrupt return logic with the Mutex release, GeMRTOS minimizes the overhead and potential race conditions that could arise if these operations were performed separately.

The ability to atomically return from an interrupt while releasing the Mutex is a key design feature that enhances the responsiveness and determinism of the GeMRTOS system, particularly in real-time and safety-critical applications.

4.2.3 Status Register (STATUS)

The Status Register (STATUS) provides visibility into the internal state of the GeMRTOS controller. It reports the last event that occurred, the current time prescale, and the status of the time counters and frozen mode. This information is crucial for the software to monitor the overall health and behavior of the real-time operating system.

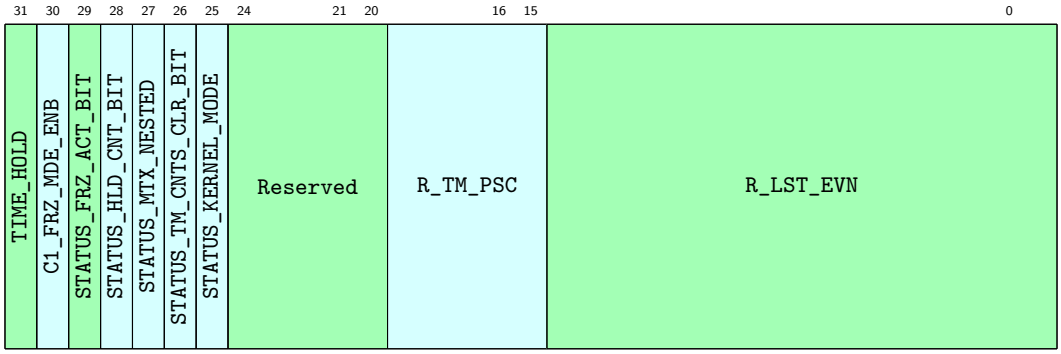

Figure 4.2: Status Register fields (STATUS)

The Status Register (STATUS) returns the following information when read (Fig. 4.2):

- **Bits 15:0** (R_LST_EVN): Last Event Occurred (LST_EVN_OCC) with the following priorities:
  - EVN_CODE_FROZEN: the GeMRTOS controller indicates that it has switched to frozen mode.
  - EVN_CODE_UNFROZEN: the GeMRTOS controller indicates that it has switched to non-frozen mode.

  - EVN_CODE_TIMED: a timed event happened because the system time counter reached the next time event.
  - DIRQi_CODE: the IRQ i took place. The trigger release associated is released.
- **Bits 20:16** (R_TM_PSC): Current time prescale. It is the value defined by setting bit 14 in the control register.
- **Bit 25** (STATUS_KERNEL_MODE): Determines if the GeMRTOS controller is in kernel mode. In kernel mode, requests for Critical Section are not nested.
- **Bit 26** (STATUS_TM_CNTS_CLR_BIT): Determines if time counter of the GeMRTOS controller are reset and hold.
- **Bit 27** (STATUS_MTX_NESTED): Determines if granted to Critical Section is nested.
- **Bit 28** (STATUS_HLD_CNT_BIT): Status of the reset of the time counters: When 1, the time counters are reset and hold. The time counters are reset and hold reset, when execution gm_TimeCountersReset(). The time counters are release to run again when executing gm_TimeCoutersUnreset().
- **Bit 29** (STATUS_FRZ_ACT_BIT): Frozen mode active. This bit is 1 if the GeMRTOS controller is in frozen mode, and 0 if it is in non-frozen mode.
- **Bit 30** (STATUS_FRZ_ENB_BIT): Frozen mode enable. This bit is 1 if frozen mode is enabled, 0 if it is disabled. The execution of gm_FrozenModeEnable() enables the frozen mode. The execution of gm_FrozenModeDisable() disables the frozen mode.
- **Bit 31** (STATUS_TIME_HOLD_BIT): This bit is status of the TIME_HOLD bit (1 time is held, 0 time is running). The execution of gm_TimeCountersHold() holds the time counters in its current state. The execution of gm_TimeCountersUnhold() unholds the time counters.

4.2.4 Control Register (CTRL)

The Control Register (CTRL) provides a comprehensive set of configuration and control options for the GeMRTOS controller. It allows the software to precisely manage the time prescale, interrupt handling, task scheduling, mutex management, and various operational modes of the GeMRTOS system. Figure 4.3 shows all bit of the Control Register.

Figure 4.3: Control Register fields (CTRL)

The Control Register (CTRL) is accessed when writing. All the bits are cleared after the command is executed. The command is completed when the bit is set. The meaning of the bits is:

- **Bits 7:0** (DATA): Data for specific operation. This data is used according to the operation in the other command bits.
- **Bit 15** (CTRL_HLT_PRC_MTX): If Bit 0 is equal to 1, the GeMRTOS halts processors when Mutex is granted for debugging. If Bit 0 is equal to 0, the Mutex is released and processors are unhalted. Only for internal use.

- **Bit 16** (CTRL_KERNEL_MODE_BIT): If Bit 0 is equal to 1, the GeMRTOS controller is set to kernel mode, otherwise is set to non-kernel mode. The gm_KernelModeSet() function sets the GeMRTOS controller in kernel mode, whilst the gm_KernelModeReset() function sets the GeMRTOS controller in non-kernel mode.
- **Bit 17** (CTRL_TM_PSC_BIT): Set the time prescale register (R_TM_PSC using the data bits 7:0. The time prescale is often configured to obtain a 10MHz time counter from the system clock frequency. The gm_SystemTimePrescaleSet function sets the system clock prescale to get the system time unit. The gm_SystemTimePrescaleGet() function returns the current time prescale.
- **Bit 18** (CTRL_LOW_PRC_BIT): Set the lowest priority processor detailed in bits 7:0. The kernel gk_SetLo westProcessor function sets the lowest priority bit.
- **Bit 19** (CTRL_IRQ_ENB_SET_BIT): Set the IRQ enable bit in the R_IRQ_ENB register of the GeMRTOS controller, enabling the IRQ of the IRQ detailed in bits 7:0. The gm_IrqEnable(irq) function is used to enable the irq external interrupt in the GeMRTOS controller. The gm_IrqDisable(irq) function disables the irq external interrupt in the GeMRTOS controller.
- **Bit 20** (CTRL_IRQ_ENB_CLR_BIT): Clear the IRQ enable bit in the R_IRQ_ENB register of the GeMRTOS controller, disabling the IRQ of the IRQ detailed in bits 7:0. The gm_IrqDisable(irq) function is used to disable the irq interrupt in the GeMRTOS controller. The gm_IrqEnable(irq) function enables the irq external interrupt in the GeMRTOS controller.
- **Bit 21** (CTRL_NTX_TM_PRC_BIT): Set the processor executing the next timed event detailed in bits 7:0 (R_NXT_TM_EVN_PRC register of the GeMRTOS controller). gk_NextOccTimeProcessor() is used by the kernel to set the processor executing the next time occurence task (0 if next time occurrence task is not executing.
- **Bit 22** (CTRL_TRG_INT_PRC_BIT): Trigger processor interrupt defined in bits 7:0. The gm_TriggerPrcI RQ(proc) produces an interrupt to processor proc.
- **Bit 23** (CTRL_ERET_ADDR_BIT): Set the interrupt return (R_ERET_REG) address specified in bits 7:0. When such an address is read, it means the processor is returning from interrupt and consequently the GeMRTOS controller Mutex is released if it is not nested. The gm_ERETAddressSet function is used by the kernel to configure this register.
- **Bit 24** (CTRL_MTX_RLS_DSB_BIT): If Bit 0 is equal to 0, enable GemRTOS controller Mutex to be released next time the Mutex is released. It is used to implement nested Mutex requests by GeMRTOS. If Bit 0 is equal to 1, disable Mutex to be released next time the Mutex is released. It is used to implement nested Mutex requests by GeMRTOS. The gm_MutexNestedValue function sets the GeMRTOS controller Mutex nested parameter according to the value.
- **Bit 25** (CTRL_FRZ_MDE_ENB_BIT): If Bit 0 is equal to 0, disable frozen mode. When frozen mode is disabled, the GeMRTOS controller remains in non-frozen mode even though the system time overcame the next event by the frozen threshold. The gm_FrozenModeDisable() function disables the frozen mode. The gm_FrozenModeEnable() function enables the frozen mode event. If Bit 0 is equal to 1, enable frozen mode. The gm_FrozenModeEnable() function enables the frozen mode event. The gm_FrozenModeDisable() function disables the frozen mode.
- **Bit 26** (CTRL_UNFRZ_MDE_CLR_BIT): UnFrozen mode event flag clear (C1_UNFRZ_EVN). When the GeMRTOS controller indicates an unfrozen mode event, this bit should be set to acknowledge the indication and continue with the next events. The gm_UnFrozenEventClear() function clears the C1_UNFRZ_EVN flags. It should be executed when the event is active in order to allow others event to be informed by the GeMRTOS controller. Is is executed only by the GeMRTOS kernel.
- **Bit 27** (CTRL_FRZ_MDE_CLR_BIT): Frozen mode event flag clear (C1_FRZ_EVN). When the GeMRTOS controller indicates a frozen mode event, this bit should be set to acknowledge the indication and continue with the next events. The gm_FrozenEventClear() function clears the frozen flag (C1_FRZ_EVN). It

should be executed when the event is active in order to allow others event to be informed by the GeMRTOS controller. Is is executed only by the GeMRTOS kernel.

- **Bit 28** (`CTRL_TM_CNTS_CLR_BIT`): If Bit 0 is equal to 1, reset and keep reset the frozen and non-frozen time counters. If Bit 0 is equal to 0, set running the frozen and non-frozen time counters. The `gm_TimeCountersReset()` function is executed at system start-up before executing `gm_TimeCoutersUnreset()`.
- **Bit 29** (`CTRL_TM_CNTS_HLD_BIT`): If Bit 0 is equal to 0, unhold the GeMRTOS time counters. The `gm_TimeCountersUnhold()` function is used to unhold the GeMRTOS time counters of the GeMRTOS controller. If Bit 0 is equal to 1, hold the GeMRTOS time counters. All time counts in the GeMRTOS controller are held at the current values. The `gm_TimeCountersHold()` function is used to hold the GeMRTOS time counters of the GeMRTOS controller.
- **Bit 30** (`CTRL_GTD_CLK_BIT`): If Bit 0 is equal to 0, disable the output clock port of the GeMRTOS controller. The output clock port turns to low when disabled. The `gm_GatedClockDisable()` function is used to disable the gated clock output of the GeMRTOS controller. If Bit 0 is equal to 1, enable the output clock port of the GeMRTOS controller. The output clock port is the GeMRTOS controller input clock when enabled. The `gm_GatedClockEnable()` function is used to enable the gated clock output of the GeMRTOS controller.
- **Bit 31** (`CTRL_RESET_BIT`): Reset the GeMRTOS controller. By writing a 1 to this bit, all the internal registers of the GeMRTOS controller are reset to their initial state. The `gm_SystemReset()` function is used to set this control bit.

Together, the CTRL and STATUS registers form a powerful interface for the GeMRTOS software to interact with the underlying hardware controller, enabling fine-grained control and observability of the real-time execution environment.

4.2.5 `PRC_IRQ_SECTION` Register

The `PRC_IRQ_SECTION` register returns the state of the processors when they are in the IRQ section of GeMRTOS. When a processor interrupts another processor by triggering a hardware interrupt, it must wait until the corresponding bit in this register is set in order to be sure the interrupt was successful.

The use of this register ensures that inter-processor interrupts are handled in a deterministic and reliable manner, preventing race conditions and guaranteeing the successful delivery of interrupts. This is a key feature for maintaining the real-time responsiveness and synchronization of the GeMRTOS system in a multiprocessor environment.

The information provided by the `PRC_IRQ_SECTION` register is crucial for the GeMRTOS software to coordinate the execution of interrupt service routines across the various processors, enabling efficient and robust interrupt handling throughout the system.

4.2.6 `SYS_TM_HGH` Register

The `SYS_TM_HGH` register holds the most significant word of the system time register. This register, along with the `SMP` register, forms the complete 64-bit system time counter maintained by the GeMRTOS controller.

After reading the `SYS_TM_HGH` register, the software should also read the `SMP` register to obtain the lowest significant word of the system time. Together, these two registers provide the full 64-bit system time value.

The System Time register represents the total elapsed time tracked by the GeMRTOS controller. It is the sum of the non-frozen time and the frozen time, providing a comprehensive measure of the total time elapsed in the system.

This System Time value is crucial for the real-time scheduling and event management within the GeMRTOS framework. The software can use the system time to schedule tasks, measure execution durations, and coordinate time-sensitive operations across the system.

By maintaining a 64-bit system time counter, the GeMRTOS controller ensures that the time representation has sufficient resolution and range to accommodate the requirements of even the most demanding real-time applications.

The SYS_TM_HGH and SMP registers provide a reliable and efficient mechanism for the software to access the comprehensive system time information managed by the GeMRTOS controller.

4.2.7 FRZ_THR Register

The FRZ_THR register holds the most significant word of the frozen threshold value. This register, along with the SMP register, forms the complete 64-bit frozen threshold value used by the GeMRTOS controller.

The frozen threshold represents the maximum allowed duration for the system to be in the non-frozen mode. When the system time exceeds this threshold from the last time event missed, the GeMRTOS controller will automatically exit the non-frozen mode.

To set or read the full 64-bit frozen threshold value, the software should use the FRZ_THR register to access the most significant word, and the SMP register to access the least significant word.

This two-part register structure allows the GeMRTOS software to precisely configure and monitor the frozen threshold, which is a critical parameter for maintaining the real-time responsiveness and consistency of the system.

By providing access to the complete 64-bit frozen threshold value, the FRZ_THR and SMP registers enable the GeMRTOS software to adapt the system's behavior to the specific requirements of the target application, ensuring optimal real-time performance and fault tolerance.

4.2.8 FRZ_TM Register

The FRZ_TM register holds the most significant word of the frozen time value maintained by the GeMRTOS controller. This register, along with the SMP register, represents the complete 64-bit frozen time counter.

The frozen time is a measure of the total duration that the GeMRTOS system has spent in the frozen mode. This value is accumulated whenever the controller enters the frozen mode, which is typically used to temporarily suspend the real-time execution in response to certain events or conditions.

To read the full 64-bit frozen time value, the software should first access the FRZ_TM register to obtain the most significant word, and then read the SMP register to get the least significant word.

By providing access to the complete frozen time information, the GeMRTOS software can monitor and analyze the system's behavior in the frozen mode.

The FRZ_TM register is a crucial component of the GeMRTOS time management infrastructure, allowing the software to maintain a comprehensive understanding of the system's temporal behavior and adapt its strategies accordingly.

4.2.9 TM_CNT Register

The TM_CNT register holds the most significant word of the non-frozen time counter maintained by the GeMRTOS controller. This register, along with the TIME_LSW register, represents the complete 64-bit non-frozen time value.

The non-frozen time is a measure of the total elapsed time that the GeMRTOS system has spent in the non-frozen mode. This value is continuously updated by the controller and is the basis for all time-related operations, such as task scheduling, timeouts, and time-stamping of events.

To read the full 64-bit non-frozen time value, the software should first access the TM_CNT register to obtain the most significant word, and then read the SMP register to get the least significant word.

The non-frozen time value is a crucial component of the GeMRTOS time management system. It represents the "wall clock" time that is valid for all timed events and tasks within the real-time operating system. This information is essential for the software to coordinate the execution of real-time applications and ensure that timing constraints are met.

By providing access to the complete non-frozen time information through the TM_CNT and SMP registers, the GeMRTOS software can accurately track the passage of time, schedule tasks, and measure the performance of the system, even in the presence of frozen mode operations.

This time management functionality is a cornerstone of the GeMRTOS real-time capabilities, enabling the system to maintain deterministic and predictable behavior in complex, time-critical applications.

4.2.10 NXT_OCC_TM Register

The NXT_OCC_TM register is used to set the most significant word of the next occurrence time in the GeMRTOS controller. This register, in conjunction with the SMP register, allows the software to specify the complete 64-bit value for the next scheduled event or task occurrence.

The next occurrence time is a critical parameter in the GeMRTOS scheduling system. It allows the software to precisely specify when the next scheduled event should occur, enabling fine-grained control over the timing of real-time operations.

By providing a 64-bit value for the next occurrence time, GeMRTOS ensures that the system can handle a wide range of timing requirements.

This register plays a crucial role in maintaining the deterministic behavior of the real-time system, allowing for accurate and reliable scheduling of tasks and events across the entire GeMRTOS environment.

4.2.11 ELAPSE_TM Register

The ELAPSE_TM register is a 32-bit time counter that provides a mechanism for measuring elapsed time between two consecutive read operations. This register has a unique behavior that makes it particularly useful for performance monitoring and time interval measurements in the GeMRTOS system.

Key features of the ELAPSE_TM register:

- 32-bit counter: Provides a wide range for time measurements.
- Auto-reset on read: The counter automatically resets to zero when it is read.
- Measures time between reads: Returns the time elapsed since the last read operation.

This behavior makes the ELAPSE_TM register an invaluable tool for various timing-related operations in the GeMRTOS environment.

By providing a simple yet effective means of measuring time intervals, the ELAPSE_TM register enhances the capabilities of GeMRTOS for precise timing control and performance analysis, contributing to the overall efficiency and reliability of real-time applications running on the system.

4.2.12 INOUTS Register

The INOUTS register serves as a versatile interface for handling both inputs and outputs of the GeMRTOS controller. This dual-purpose register provides a convenient mechanism for the software to interact with external devices and signals, facilitating efficient I/O operations within the GeMRTOS environment.

Key features of the IO register:

- Input and Outputs: Functions as both an input and output register.
- Direct access: Allows easy reading of inputs and writing of outputs.
- Simplified I/O handling: Enables straightforward management of I/O operations from within programs.

Functionality:

1. Input operations:
   - Reading the INOUTS register returns the current state of the input pins.
   - Allows the software to quickly sample external signals or sensor data.
2. Output operations:
   - Writing to the INOUTS register sets the state of the output pins.
   - Enables the software to control external devices or signal other components.

The INOUTS register plays a crucial role in the GeMRTOS system by:

- Simplifying I/O programming: Provides a unified interface for both input and output operations.
- Enhancing real-time responsiveness: Allows quick access to I/O states without complex peripheral configurations.
- Facilitating hardware abstraction: Enables software to interact with external hardware through a standardized register.
- Supporting diverse applications: Accommodates a wide range of I/O requirements for various real-time systems.

By incorporating the INOUTS register, GeMRTOS offers developers a powerful and flexible tool for managing input and output operations efficiently. This capability is essential for creating responsive and interactive real-time applications that can effectively interface with the external environment.

4.2.13 RST_CNT Register

The RST_CNT register is a crucial component in the GeMRTOS controller that facilitates the coordinated and sequential start-up of multiple processors in a multi-core system. This register provides essential information for synchronizing the initialization process across different processor cores.

Key features of the RST_CNT register:

- Returns the ID of the last processor unreset
- Enables sequential processor start-up
- Supports synchronization in multi-core systems

The RST_CNT register plays a vital role in the GeMRTOS system by:

- Ensuring orderly system initialization: Helps maintain a predictable and controlled start-up sequence.

- Facilitating inter-processor coordination: Allows each processor to be aware of the system's overall initialization state.
- Supporting scalability: Enables GeMRTOS to efficiently manage systems with varying numbers of processor cores.
- Enhancing system reliability: Reduces the risk of initialization-related issues in multi-processor environments.

By incorporating the RST_CNT register, GeMRTOS provides a robust mechanism for managing the complex process of initializing multi-processor systems. This feature is essential for ensuring that all processor in the system start up in a coordinated and efficient manner, laying the foundation for reliable and deterministic operation of real-time applications in multi-core environments.

4.2.14 SMP Register

The SMP register is a crucial component in the GeMRTOS controller that facilitates 64-bit data transfers. This register works in conjunction with other registers to enable reading and writing of 64-bit values across the system.

Key features of the SMP register:

- Support for 64-bit data transfers
- Acts as a temporary storage for least significant words
- Coordinates with other registers for complete 64-bit operations

Functionality:

1. Reading 64-bit values:
   - First, read the most significant word (MSW) from the appropriate register.
   - Then, read the SMP register to obtain the least significant word (LSW).
2. Writing 64-bit values:
   - First, write the least significant word (LSW) to the SMP register.
   - Then, write the most significant word (MSW) to the appropriate register.

The SMP register plays a vital role in the GeMRTOS system by:

- Enabling precise 64-bit operations: Allows for accurate representation and manipulation of large values or time stamps.
- Maintaining data integrity: Ensures that 64-bit values are read or written atomically.
- Simplifying software implementation: Provides a consistent mechanism for handling 64-bit transfers across different registers.

By incorporating the SMP register, GeMRTOS provides a robust and efficient mechanism for handling 64-bit data transfers. This capability is essential for maintaining high precision in time-critical operations and supporting applications that require extended range values in real-time systems.

4.2.15 MTX_GRN (Mutex Grant) Register

The MTX_GRN (Mutex Grant) register is a crucial component in the GeMRTOS controller that manages mutex (mutual exclusion) allocation across multiple processors in a multi-core system. This register provides a bitmap representation of mutex ownership, allowing for efficient synchronization and resource sharing among processors.

The MTX_GRN register plays a vital role in the GeMRTOS system by:

- Ensuring data integrity: Prevents simultaneous access to shared resources by multiple processors.
- Facilitating inter-processor communication: Provides a hardware-level mechanism for synchronization.
- Enhancing system performance: Allows for quick mutex status checks without software overhead.
- Supporting scalability: Can accommodate systems with varying numbers of processor cores.

By incorporating the MTX_GRN register, GeMRTOS provides a robust and efficient mechanism for managing mutex

grants in multi-processor systems. This feature is essential for ensuring proper synchronization and resource sharing among multiple processors, contributing to the overall reliability and performance of real-time applications in complex, multi-processsor environments.

This overview of the GeMRTOS global region provides a foundation for understanding the core control mechanisms within the system. The subsequent section will delve into the specifics of the processor region, detailing the registers that enable tailored management of individual processors.

4.3 GeMRTOS Processor Region

The GeMRTOS Processor Region is a crucial component of the GeMRTOS Controller architecture, providing individual control and status registers for each processor in a multi-processor system. This region facilitates efficient management and monitoring of processor-specific operations within the GeMRTOS environment.

Key features of the GeMRTOS Processor Region:

- Individual registers for each processor
- Accessible through individual memory-mapped bus for each processor
- Enables fine-grained control and status monitoring avoiding race conditions

The registers in the GeMRTOS Processor Region are organized based on address offsets from the `GEMRTOS_DRIVER_PROCESSOR_BASE` memory address. This structure allows for consistent and predictable access to processor-specific registers across different system configurations.

Table 4.2 shows the registers and their corresponding address offsets within the GeMRTOS Processor Region.

Table 4.2: GeMRTOS Processor Region Registers.

Dword Offset	Register Name	R/W	Description
0x00	`ADDR_REG_0`	R/W	The `ADDR_REG_0` register manages processor states, mutex grants, and interrupt requests in the GeMRTOS system. Its behavior differs between read and write operations.
0x01	`ADDR_REG_1`	R/W	The `ADDR_REG_1` register allows for fine-grained management of processor interrupts, mutex operations, and processor states.
0x02	`ADDR_REG_2`	R/W	The `ADDR_REG_2` register allows for fine-grained management of disabling processor states.

The GeMRTOS Processor Region plays a vital role in the overall system by providing several key benefits:

- `CPU_ID Assignment:`
  - Assigns a unique `CPU_ID` for each processor without requiring special registers in processor architectures.
  - Simplifies processor identification and management across diverse hardware platforms.
- **Scalable Multi-Processor Support:**
  - Facilitates efficient management of systems with parameterizable numbers of processors.
  - Enables seamless scaling from uniprocessor to multi-processor configurations without major software modifications.

These features collectively contribute to the robustness and versatility of the GeMRTOS system:

1. **Hardware Abstraction:** By providing a consistent method for CPU identification and management, GeMRTOS simplifies porting to different processor architectures.

2. **Scalability:** The system can easily adapt to different multi-processor configurations, supporting a wide range of embedded and real-time applications.
3. **Customization:** Developers can fine-tune the behavior of individual processors to meet specific performance or functionality requirements.

By providing a dedicated region for processor-specific registers, GeMRTOS offers a powerful and flexible framework for developing and managing complex real-time systems in multi-processor environments.

4.3.1 ADDDR_REG_0 Register

This register manages processor states, mutex grants, and interrupt requests in the GeMRTOS system. Its behavior differs between read and write operations.

Write Operation: When writing to this register:

- If the PRC_HLT_ENB_BIT (Processor Halt Enable) bit is set, the processor enters a wait state when writing the ADDR_REG_0 address in the processor memory region. The processor exits from wait when the GemRTOS controller Mutex is granted to the processor or an interrupt request is issued to the processor. The gm_PrcInterruptEnable() function sets the halt bit.
- The processor is released from this wait state when either:
  - The processor is granted the GeMRTOS Mutex, or
  - The processor receives an interrupt request.

This mechanism allows for efficient processor synchronization and power management by halting the processor until specific conditions are met.

Read Operation: When reading from this register, the following information is available (Figure 4.4):

- **Bit 0:** PRC_HLT_ENB_BIT **State**
  - Indicates the current state of the Processor Halt Enable for this processor.
  - 0: Processor Halt is disabled
  - 1: Processor Halt is enabled
- **Bit 1 (**R_MTX_PRC_GRN**): GeMRTOS Mutex Grant State**
  - Indicates whether this processor has been granted the GeMRTOS Mutex.
  - 0: Mutex not granted to this processor
  - 1: Mutex granted to this processor
- **Bit 2 (**R_PRC_INT**): Processor Interrupt Request**
  - Indicates whether an interrupt request is pending for this processor.
  - 0: No interrupt request pending
  - 1: Interrupt request pending

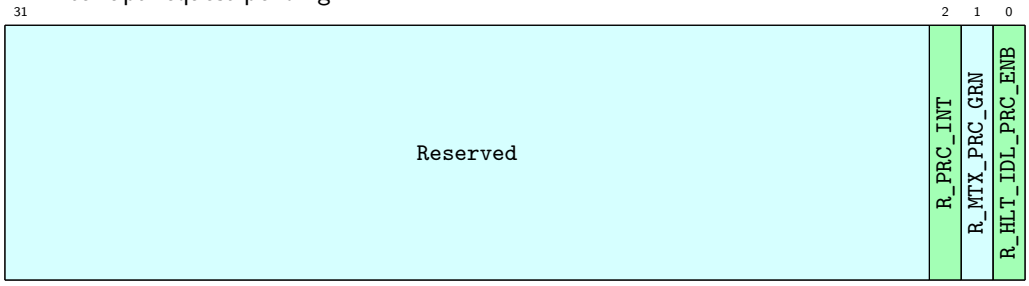

Figure 4.4: Read ADDDR_REG_0 Register

Functionality: This register serves several crucial functions in the GeMRTOS system:

1. **Processor State Control:** Allows for fine-grained control over processor execution states, enabling power-saving modes when the processor is idle.
2. **Synchronization:** Facilitates inter-processor synchronization through GeMRTOS Mutex management, crucial for maintaining data integrity in multi-processor systems.
3. **Interrupt Management:** Provides a mechanism for monitoring and controlling interrupt requests, essential for real-time responsiveness.
4. **Status Monitoring:** Enables software to quickly assess the current state of the processor, including its halt status, GeMRTOS Mutex ownership, and pending interrupts.

4.3.2 `ADDR_REG_1` Register

The `ADDR_REG_1` register is a crucial component in the GeMRTOS controller, providing control over various processor-specific functions. This register allows for fine-grained management of processor interrupts, mutex operations, and processor states. Figure 4.5 shows the `ADDR_REG_1` bit assignment.

Register Bit Assignments:

- **Bit 0 (`PRC_INT_ENB_BIT`): Processor Interrupt Enable**
  - When set, enables interrupts for the processor.
  - Allows the processor to respond to interrupt requests from the GeMRTOS controller.
- **Bit 1 (`PRC_MTX_RSV_BIT`): Mutex Reservation**
  - When set, requests the GeMRTOS controller to grant the mutex to this processor.
  - Used for synchronization in multi-processor operations.
- **Bit 2 (`PRC_IRQ_ACK_BIT`): Interrupt Request Acknowledgement**
  - When set (value=1), clears/acknowledges the pending interrupt request. This is the "acknowledge" operation that removes the IRQ from pending state.
  - Should be set when the processor finishes interrupt service by the `gm_PrcIRQSecSet()` function.
- **Bit 3 (`PRC_IDL_ENB_BIT`): Idle Waiting Enable**
  - When set, enables the processor to enter a wait state upon writing to `ADDR_REG_0`.
  - Facilitates power-saving modes and efficient processor management.
- **Bit 4 (`PRC_MTX_RLS_BIT`): Mutex Release**
  - When set, releases the GeMRTOS controller mutex previously granted to the processor.
  - Allows other processors to acquire the mutex.

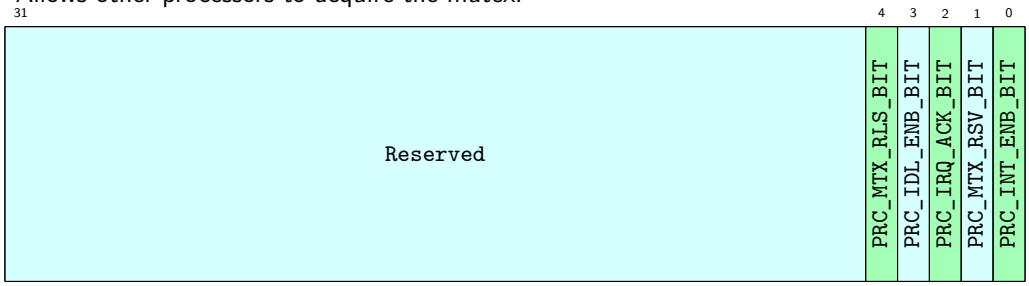

Figure 4.5: ADDR_REG_1 Register

Functionality: The `ADDR_REG_1` register provides essential control mechanisms for:

1. **Interrupt Management:** Enabling/disabling interrupts and acknowledging interrupt requests.
2. **Mutex Control:** Requesting and releasing the GeMRTOS controller mutex for inter-processor synchronization.
3. **Power Management:** Controlling processor idle states for energy efficiency.

The ADDR_REG_1 register plays a vital role in the GeMRTOS system by providing a centralized mechanism for controlling key aspects of processor behavior.

4.3.3 ADDR_REG_2 Register

The ADDR_REG_2 register provides access to disable the processor interrupt in the GeMRTOS controller. Figure 4.6 shows the ADDR_REG_2 bit assignment.

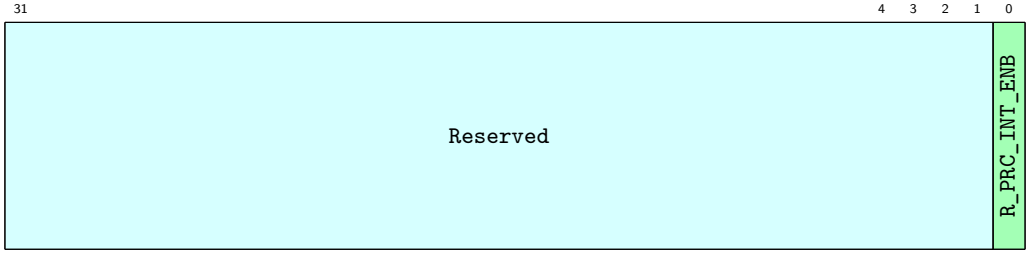

Figure 4.6: ADDR_REG_2 Register

Register Bit Assignments:

- **Bit 0 R_PRC_INT_ENB: Processor Interrupt Enable (PRC_INT_ENB)**
  - When set, disables interrupts for the processor.

In summary, the GeMRTOS Processor Region is a cornerstone of the system's architecture, providing essential functionality for multi-processor management, system customization, and debugging. This design choice enhances the overall flexibility, scalability, and ease of use of GeMRTOS in a wide variety of real-time and embedded applications.

4.4 GeMRTOS Controller Mutex Architecture

The GeMRTOS Controller implements a shared resource access control mechanism through a mutex system. This system allows the different processors in the system to coordinate access to critical resources, avoiding race conditions and ensuring data integrity.

4.4.1 Mutex Structure

The GeMRTOS Controller mutex is composed of the following key elements:

Mutex Reservation Register: This is the bit 1 of the ADDR_REG_1 register in processor memory region of the GeMRTOS controller. Setting this bit allows the processor to request ownership of the mutex.

Mutex Grant Indicator: This is the bit 1 in the processor memory region of the GeMRTOS controller, which indicates whether the current processor has the mutex granted.

Mutex Release Mechanism: When a processor has completed its critical operations, it can release the mutex by setting the bit 4 in the ADDR_REG_1 register in the processor memory region of the GeMRTOS controller.

4.4.2 Mutex Acquisition

For a processor to acquire the GeMRTOS Mutex, it must follow these steps:

1. Set the mutex reservation bit (bit 1) in the ADDR_REG_1 register in the processor memory region of the GeMRTOS controller.

2. Verify the mutex grant indicator (bit 1) in the ADDR_REG_1 register in the processor memory region of the GeMRTOS controller.
   - If the mutex is granted, the processor can proceed with its critical operations.
   - If the mutex is not granted, the processor may enter a wait state (HALT or infinite loop) from which the GeMRTOS controller will wake it up when the GeMRTOS Mutex is granted or when the processor is interrupted.

The wait state is triggered by setting the bit 0 in the ADDR_REG_0 register of the processor memory region of the GeMRTOS controller. This bit activates the waitrequest signal to put the processor in a wait state, which is then cleared when the GeMRTOS Mutex is granted or when the processor's interrupt is activated.

4.4.3 Mutex Release

When a processor has completed its critical operations, it must release the GeMRTOS Mutex to allow other processors to access the shared resource:

The processor can release the mutex in two ways:

- Set the bit 5 in the ADDR_REG_1 register of the processor memory region of the GeMRTOS controller to clear the GeMRTOS Mutex register.
- Perform a return intruction from PRC_CDE_ADDRx register address in the global memory region of the GeMRTOS controller, the to execute a new task.

In both cases, the release of the mutex must be enabled by setting the bit 18 in the STATUS_CTRL register of the GeMRTOS controller's global region. If this bit 18 is not active, the GeMRTOS Mutex will not be released. This is used to allow the nesting of GeMRTOS Mutex requests.

This GeMRTOS Controller mutex mechanism allows processors to coordinate access to shared resources efficiently and reliably, avoiding race conditions and ensuring data integrity in multi-processor environments, even with nested mutex requests.

4.4.4 Mutex Release Decision Tree

The GeMRTOS controller implements a sophisticated 5-way multiplexer that determines which processor holds the mutex on each clock cycle. This decision tree ensures that the mutex is released only when appropriate, considering system state, processor mode, and task nesting level.

Release Decision Precedence

The hardware evaluates the following conditions in order (highest to lowest priority):

1. **Frozen Mode Release:** If system enters frozen mode AND no current holder exists, release all processors.
2. **Grant to Waiter:** If no processor currently holds the mutex, grant it to the next waiting processor (using round-robin priority).
3. **Nested Flag Gate (Critical):** If R1_MTX_NESTED $= 1$, HOLD the mutex with the current holder. This gate completely blocks release regardless of other signal states. This is the core mechanism that enforces nested critical section semantics.
4. **Kernel Mode Protection:** If processor is in kernel mode, HOLD the mutex to protect kernel operations.
5. **Return From Exception:** If ERET address is detected (return from interrupt), RELEASE the mutex unconditionally.

6. **Default (Conditional Release):** If no higher-priority condition applies, apply release bits: only processors with release-bit set will release.

Critical: The Nested Flag Gate

Condition 3 above is the most important for nested critical sections. When `R1_MTX_NESTED` $= 1$, the entire release logic is blocked. This means:

- Release bit has NO effect when nested flag is set
- Processor holds mutex regardless of any other signals
- Only when nested flag becomes 0 can release occur

This implements the nested lock semantics: a processor can only release the mutex when the task has exited ALL nested critical sections (nested counter reaches 0).

Register Reference

Table 4.3: Mutex Register Reference.

Register	Bits	Name	Purpose
R1_MTX_NESTED	[0]	Nested Flag	Controls release gate (0=can release, 1=hold)
R_MTX_PRC_GRN	[N-1:0]	Grant	One bit per processor; 1=holds mutex
R_MTX_RLS_BIT	[N-1:0]	Release Requests	One bit per processor; set to request release

4.4.5 R1_MTX_NESTED Synchronization Protocol

The nested flag (`R1_MTX_NESTED`) must be kept in synchronization with the current task's nesting counter (`TCB_MTX_NESTED`). This section describes the synchronization mechanism and protocol.

Two-Level Nesting Model

GeMRTOS uses two separate nesting counters at different levels:

Software Level (TCB_MTX_NESTED): Tracks the actual nesting depth of the current task.

- Location: Task Control Block (GS_TCB structure)
- Type: Unsigned integer (32-bit)
- Range: 0 to 2^{32}-1
- Semantics: Incremented on entry to each nested critical section, decremented on exit
- Example: 0 = task not in CS, 1 = in first level, 2 = in second level, etc.

Hardware Level (R1_MTX_NESTED): Binary flag that gates mutex release.

- Location: GeMRTOS controller register

- Type: 1-bit flag (not a counter)
- Range: 0 or 1 only
- Semantics: 0 = processor can release mutex, 1 = processor must hold mutex
- Relationship: $\texttt{R1_MTX_NESTED} = (\texttt{TCB_MTX_NESTED} == 0)?0 : 1$

Synchronization Points

The nested flag must be synchronized at two specific points:

1. **Point A - Mutex Acquisition (Automatic in Hardware):**
   - When processor acquires mutex via ADDR_REG_0 write
   - Hardware automatically reads current task's TCB_MTX_NESTED
   - Hardware automatically loads: $\texttt{R1_MTX_NESTED} \leftarrow (\texttt{TCB_MTX_NESTED} == 0?0 : 1)$
   - Timing: Same clock cycle as grant decision
   - Windows emulation must replicate this in software (see Windows Emulation chapter)
2. **Point B - Nesting Level Change (Software via IOWR):**
   - When user task enters/exits critical section
   - Software calls function: `gm_MutexNestedValue(new_nesting_value)`
   - This expands to IOWR to ADDR_CTRL_STATUS with bit 24 set
   - Hardware handler updates: $\texttt{R1_MTX_NESTED} \leftarrow \texttt{value}\&0x1$
   - Timing: Same clock cycle as IOWR completion

Protocol Invariant

At all times during mutex ownership, the following invariant MUST hold:

$$\texttt{R1_MTX_NESTED} == ((\texttt{TCB_MTX_NESTED} == 0)?0 : 1) \tag{4.1}$$

If this invariant is violated, mutex behavior becomes unpredictable and deadlocks can occur.

Protocol Sequence Example

1. Task calls `gm_GeMRTOSCriticalSectionEnter()` (level 1)
2. Writes to ADDR_REG_0 to request mutex
3. Hardware grants mutex and auto-syncs: R1_MTX_NESTED ← 0 (since TCB_MTX_NESTED still 0)
4. Task increments TCB_MTX_NESTED: 0 → 1
5. Task calls `gm_MutexNestedValue(1)` → IOWR updates R1_MTX_NESTED: 0 → 1

6. Task calls `gm_GeMRTOSCriticalSectionEnter()` (level 2 - nested)
7. No ADDR_REG_0 write (already inside CS), but TCB_MTX_NESTED incremented: 1 → 2
8. Task calls `gm_MutexNestedValue(2)` → IOWR updates R1_MTX_NESTED: 1 → 1 (stays 1)
9. Task calls `gm_GeMRTOSCriticalSectionExit()` (exit level 2)
10. Task decrements TCB_MTX_NESTED: 2 → 1
11. Task calls `gm_MutexNestedValue(1)` → IOWR: R1_MTX_NESTED stays 1
12. Task calls `GRTOS_MTX_RLS` (release request) → Hardware checks R1_MTX_NESTED=1 → HOLDS
13. Task calls `gm_GeMRTOSCriticalSectionExit()` (exit level 1)
14. Task decrements TCB_MTX_NESTED: 1 → 0
15. Task calls `gm_MutexNestedValue(0)` → IOWR: R1_MTX_NESTED: 1 → 0
16. Task calls `GRTOS_MTX_RLS` (release request) → Hardware checks R1_MTX_NESTED=0 → RELEASES

4.4.6 Nested Mutual Exclusion Terminology

To avoid confusion, this manual carefully distinguishes between two related but different concepts:

`TCB_MTX_NESTED`: The **software nesting counter** maintained by the kernel.

- Stored in each Task Control Block (TCB)
- Tracks how many nested critical sections the task has entered
- Type: Unsigned integer (can be 0, 1, 2, 3, ..., N)
- Updated by: Kernel when task enters/exits critical sections
- API: Incremented by `gm_GeMRTOSCriticalSectionEnter()`
- API: Decremented by `gm_GeMRTOSCriticalSectionExit()`
- Example values:
  - 0 = task not in any critical section
  - 1 = task in first-level critical section
  - 2 = task in second-level (nested within first-level) critical section
  - N = task in Nth-level nested critical section

`R1_MTX_NESTED`: The **hardware nested flag** in the GeMRTOS controller.

- Stored in GeMRTOS controller hardware (VHDL register)
- Binary gate that controls whether mutex can be released
- Type: Single bit (0 or 1 only - NOT a counter)

- Updated by: Hardware (automatic on acquire) or software (via IOWR)
- Read by: Release decision logic (the 5-way mux)
- Semantics:
  - 0 = processor CAN release the mutex
  - 1 = processor MUST HOLD the mutex
- Example values:
  - 0 = gate open (release permitted)
  - 1 = gate closed (release blocked)

Relationship Between the Two

The nested flag (`R1_MTX_NESTED`) is derived from the nesting counter (`TCB_MTX_NESTED`) via a simple conversion:

$$\texttt{R1_MTX_NESTED} = \begin{cases} 0 & \text{if } \texttt{TCB_MTX_NESTED} == 0 \\ 1 & \text{if } \texttt{TCB_MTX_NESTED} > 0 \end{cases} \tag{4.2}$$

This conversion compresses the full nesting depth (0, 1, 2, ..., N) into a binary gate signal (0 or 1). The hardware only needs to know "is there a nesting?" not "how deeply nested?"

Why Two Levels?

- **Software level** provides fine-grained control for complex task logic
- **Hardware level** provides simple, fast decision-making for release logic
- **Synchronization** between them ensures correctness without coupling hardware and software too tightly
- **Separation** allows hardware to be simple (one bit gate) while software is flexible (can track arbitrary nesting depth)

4.5 Time Management in the GeMRTOS Architecture

Time management within the GeMRTOS architecture is crucial for enabling real-time operation and ensuring timely execution of tasks. The system relies on a combination of hardware and software components to track time, and handle timed events with precision. This involves a timer-tickless approach, reducing overhead and improving responsiveness to critical system events. The foundation of time management is the precise control and configuration of hardware timers, allowing for granular control over time intervals.

Central to the time management strategy is the utilization of dedicated hardware registers and counters within the controller. A time prescaler allows for scaling the system clock to provide flexible time units. Up-counters track elapsed time, triggering time events upon reaching predefined values stored in specific registers. The architecture also supports "frozen mode," which provides a mechanism for handling situations where event processing lags behind real-time, allowing the system to prioritize critical operations without losing track of overall timing.

The GeMRTOS architecture utilizes specific registers to manage time events:

- R_TM_CNT register: Increments when the system is in non-frozen mode, providing a continuously updating system time.
- R_NXT_EVN_CNT register: Holds the time of the next timed event, and the hardware compares these registers to determine event occurrences.
- R_FRZ_TM_THR register: Defines a threshold for maximum time lag between an event occurrence and its processing; if this threshold is exceeded, the system enters in the frozen mode to maintain timing integrity.

Events within the GeMRTOS controller are prioritized using both timed events, configured to occur at specific times, and device interrupt requests (DIRQs), triggered by external hardware. Timed events occur when the system time counter reaches the value stored in the R_NXT_EVN_CNT register, signaling the need for time-sensitive task execution.

The GeMRTOS controller manages two primary timekeeping modes:

- **Non-frozen Mode:** Represents the cumulative time the system operates under normal conditions, with events being processed in a timely manner. GeMRTOS typically resides in this mode.
- **Frozen Mode:** The GeMRTOS controller transitions to frozen mode when timed events cannot be processed promptly, and the next timed event's occurrence time falls behind a predefined "frozen threshold," indicating a backlog.

The system's total elapsed time is the sum of time spent in both non-frozen and frozen modes. A timed event triggers when the next scheduled occurrence time is greater than or equal to the current system time. At that point, if a specific processor is designated for handling the event, that processor is interrupted; otherwise, the interrupt is issued to the lowest-priority processor to ensure event processing.

By combining hardware and software elements, and two primary timekeeping modes, the controller is an optimized and powerful system.

The following section will discuss the GeMRTOS controller device interrupt management, so you will have even better performance.

4.6 GeMRTOS Controller Device Interrupt Management

Device interrupt management is a fundamental aspect of the GeMRTOS architecture, enabling the system to respond efficiently to external events triggered by peripheral devices. This mechanism enables the system to handle asynchronous events in a timely manner without continuously polling devices, optimizing processor utilization and improving overall system responsiveness. The device interrupt management strategy integrates both hardware and software components to ensure reliable and prioritized handling of interrupt requests.

The GeMRTOS controller incorporates an interrupt input module to receive interrupt requests from various peripheral devices. These requests, denoted as Device Interrupt Requests (DIRQs), signal the need for immediate attention from the system processors. Each DIRQ can be individually enabled or disabled, allowing the system to selectively respond to specific device events based on system configuration and operational requirements.

The next actions taken on an interrupt request are:

- The controller prioritizes and manages the request, ultimately generating a Processor Interrupt Request (PIRQ) to one or more of the system's processors.
- The controller employs specific registers, such as the R_IRQ_ENB register, to enable or disable individual DIRQs and the R_IRQ_PND register to track pending interrupt requests.
- The interrupt handler section, executed by the designated processor, processes the interrupt event and takes appropriate action by releasing the associated trigger resource.

To prevent race conditions and ensure data consistency, interrupt handlers operate under critical sections, managed by the GeMRTOS Mutex, to protect shared resources.

This combination of hardware and software ensures predictable and reliable handling of device interrupts within the GeMRTOS environment. The GeMRTOS controller manages the peripheral interrupt requests by selecting the processor most adequate to serve the request from a peripheral. When a peripheral issues an interrupt request, the lowest priority processor interrupt request is asserted.

4.7 Conclusions

The GeMRTOS controller is the central element in the GeMRTOS framework, providing a foundation for real-time, multiprocessor operations. Through its efficient management of time, external events, and processor synchronization, the controller enables consistent and predictable system behavior, maximizing system capabilities for complex embedded applications.

This chapter detailed the core architectural components of the controller. These components, including specialized registers and memory regions, work in a collaborative approach. By providing a centralized point of control for these critical functions, GeMRTOS ensures robust and reliable system operation in demanding environments.

These elements also collaborate for enabling a flexible, low latency device interrupt and management tasks. The adaptable management architecture facilitates precise control over interrupt handling and processor resources to deliver well controlled operation.