GeMRTOS: Multiprocessor RTOS

User and Development Manual — Chapter 6: GeMRTOS IP for Platform Designer + Chapter 7: Hardware and Software Design Flow

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 6

GeMRTOS IP for Platform Designer

6.1 Introduction

The GeMRTOS Multiprocessor IP component simplifies the design of multiprocessor systems, offering significant advantages over traditional development methods. By reducing the complexity of processor booting, synchronization, and mutual exclusion, the IP component makes designing a multiprocessor system nearly as straightforward as creating a uniprocessor system. This streamlined approach lowers the barrier to entry for developers seeking to harness the power of parallel processing in their embedded designs.

6.2 GeMRTOS IP Architecture

The GeMRTOS IP architecture includes all the components required to implement multiprocessor systems along with other components for ease of system design and debugging. As depicted in Figure 1.1, the GeMRTOS Multiprocessor IP incorporates the following components, each designed for a specific purpose:

- **GeMRTOS Controller:** This serves as the core of the GeMRTOS Multiprocessor IP component, responsible for time management and external events. It enables the Avalon-conduit port, providing easy access to external signals.
- **System Processors:** This component supports both Nios II and Nios V soft processors, offering flexibility in processor selection.
- **Nios Monitor Component:** This serves as a debugging tool, sniffing each processor bus to detect special transaction patterns. In configurations without a debugging architecture, this component operates transparently.
- **MM-Avalon Bridge for External Devices:** This bridge facilitates the definition of different clock sources for processors and external devices, enabling efficient clock domain adaptation.
- **JTAG-UART Devices:** These devices, configurable for STDIO and STDERR consoles, streamline system design and debugging.
- **MM-Avalon Bridge for HPS Interface:** This bridge enables access to and from the HPS domain memory map. Knowledge required for use.

Overall, the GeMRTOS Multiprocessor IP component significantly reduces the challenges associated with designing complex multiprocessor systems, thereby streamlining the development process and improving efficiency.

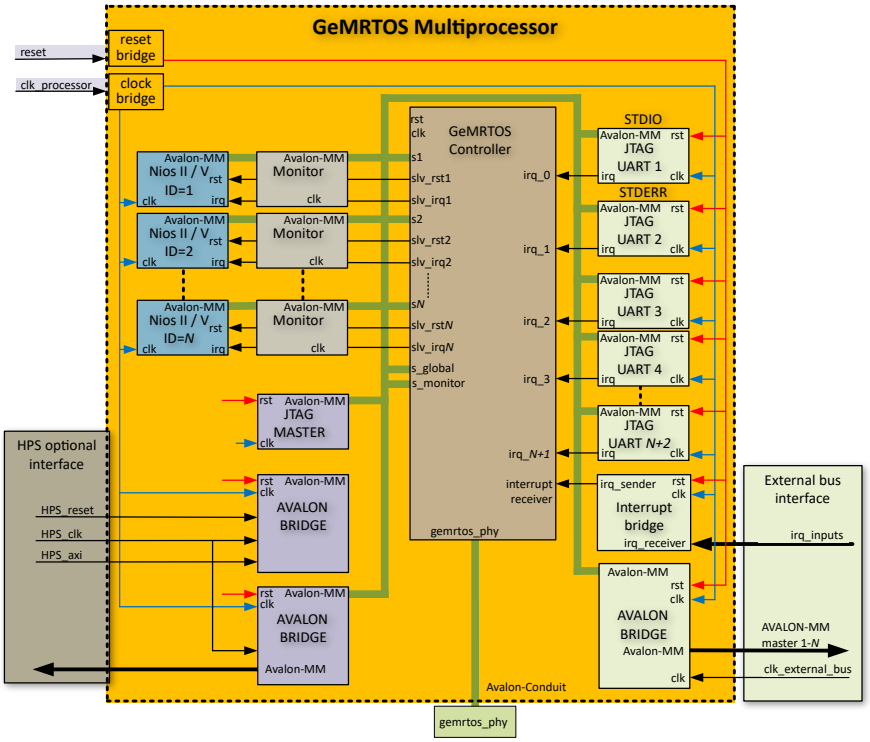

Figure 6.1: GeMRTOS Multiprocessor IP internal architecture.

6.3 GeMRTOS IP Configuration Parameters

The GeMRTOS IP component seamlessly integrates with Altera's Platform Designer development tool, appearing as the `GeMRTOS_Multiprocessor` component in the catalog. This integration provides a streamlined and adaptable approach to system configuration. Developers can tailor the GeMRTOS IP component by adjusting the parameters that affect the GeMRTOS Multiprocessor component's operation.

Table 1.1 summarizes the GeMRTOS IP configuration parameters, detailing their ranges and descriptions.

Table 6.1: GeMRTOS IP Configuration Parameters

Parameter	Range	Description
Number of processors	1-32	Determines the number of processors to be included in the GeMRTOS Multiprocessor component.
Processor type	Nios II/e, Nios II/f, Nios V/m	Determines the type of processor to be utilized.
Reset vector memory	System memories available	Determines the memory component in which the reset vector of the processor is configured. The parameter may be configured from a list of available options according to the systems memory the component is connected to.
Reset vector offset		Determines the offset in the reset vector memory for the reset vector of the processor.

Continued on next page

Table 6.1: GeMRTOS IP Configuration Parameters (Continued)

Parameter	Range	Description
Exception vector memory (only for Nios II)	System memories available	Determines the memory component in which the exception vector of the processor is configured. The parameter may be configured from a list of available options according to the systems memory the component is connected to.
Exception vector offset (only for Nios II)		Determines the offset in the exception vector memory for the exception vector of the processor.
Instruction cache size	None, 512Bytes, 1KB, 2KB, 4KB, 8KB, 16KB, 32KB, 64KB.	Set the instruction cache size for each processor of the system when it is supported.
Enable pipeline in CPU (Nios V/m only)	True / False	Enable the pipeline feature of the CPU when supported.
Independent external processor buses	True / False	When True, the GeMRTOS IP component produces one bus for each processor, useful to share the bandwidth load over several buses. When False, all the processors share the same bus, useful to simplify the design of the system. It is always advisable to use independent external processor buses to improve the performance of the system.
Include JTAG UART for STDIO	True / False	When True, a JTAG UART component is included in the GeMRTOS IP configuration for STDIO usage.
Include JTAG UART for STDERR	True / False	When True, a JTAG UART component is included in the GeMRTOS IP configuration for STDERR usage.
Include JTAG UART for each processor	True / False	When True, a JTAG UART component is included in the GeMRTOS IP configuration for each processor. This feature may be useful for debugging purposes or for demonstrations.
Enable HPS internal access	True / False	When True, a bridge is enable for accessing the GeMRTOS IP internal memory map from a HPS.
Quartus Prime project name	Selection from project folder	Determines the Quartus Prime project name in which the Platform Designer system is instantiated. This configuration is useful to automate the `gemrtos_build` script.
Platform Designer project name	Selection from project folder	Determines the name of the current Platform Designer project name, useful to automate the `gemrtos_build` script.

The GeMRTOS Multiprocessor IP parameters (illustrated in Figure 1.2) can be configured through the standard parameter windows of the Platform Designer. These configuration options become accessible when the GeMRTOS component is selected within the System Contents window of the Platform Designer interface.

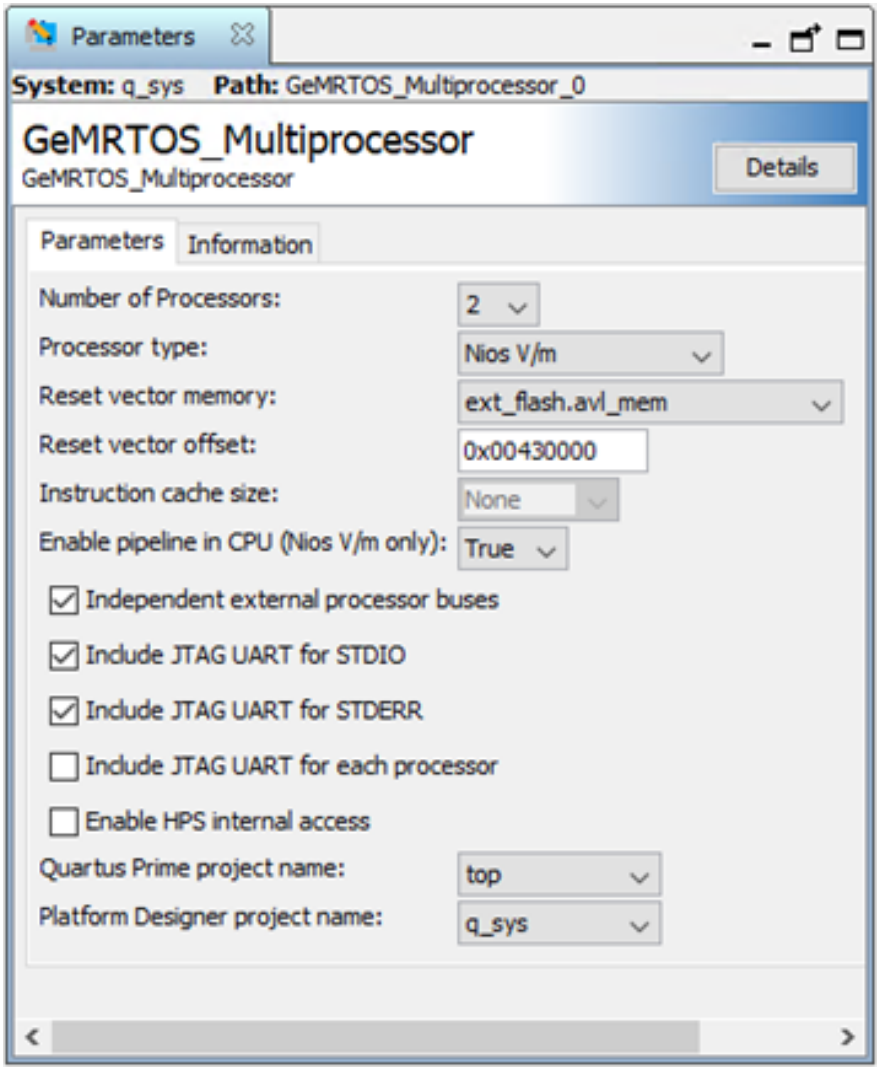

Figure 6.2: GeMRTOS IP parameter configuration tab window.

This setup allows system designers to customize the GeMRTOS Multiprocessor IP to suit their specific project requirements, integrating it effectively within the larger system architecture defined in the Platform Designer tool.

The available parameters and their possible values dynamically adapt based on two key factors:

- Connected Memory Devices: The type, number, and characteristics of memory devices connected to the GeMRTOS component influence which parameters are presented and their available options. For example:
- Different memory types (e.g., SRAM, DRAM, Flash) may expose different configuration options. The size and organization of connected memories could affect addressable ranges or caching parameters.
- Selected Processor Type: The specific processor architecture chosen for the system impacts the available features and, consequently, the relevant configuration parameters.

This adaptive parameter system ensures that:

- Users are presented with only relevant configuration options.
- The component can be optimally configured for the specific hardware environment.
- Incompatible or unavailable options are automatically hidden or disabled.

By dynamically adjusting the parameter set, the GeMRTOS configuration interface streamlines the setup process and helps prevent potential misconfigurations that could arise from presenting irrelevant options.

6.4 GeMRTOS IP Information

The GeMRTOS IP Information tab window (Figure 1.3) displays configuration information that is valuable for design and debugging purposes. It provides a quick overview of key system parameters without the need to navigate through multiple configuration windows.

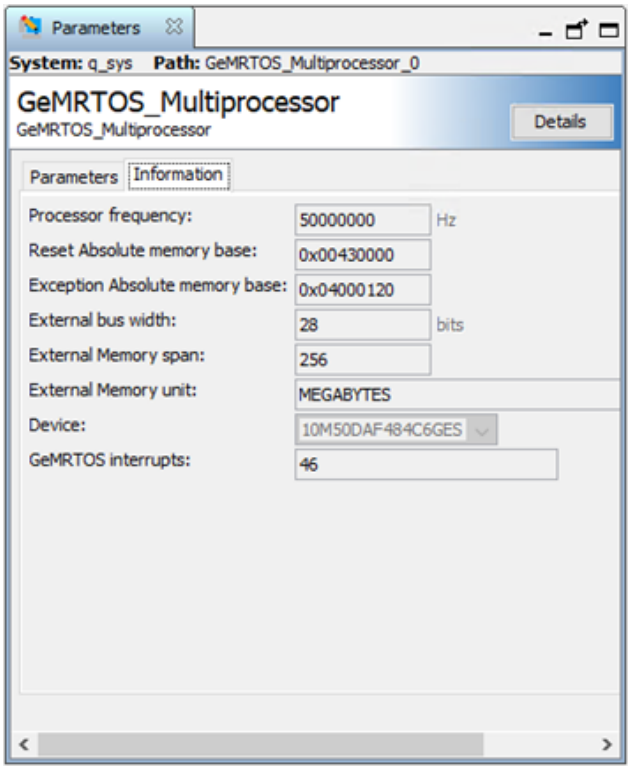

Figure 6.3: GeMRTOS IP Information tab window.

Table 6.2: GeMRTOS IP information parameters

Parameter	Description
`Processor frequency`	Shows the clock frequency connected to the processors. Important for timing calculations and performance estimations.
`Reset absolute memory base`	Displays the absolute address where processors reset. Useful when defining addresses in flash memory regions shared with FPGA configuration.
`Exception absolute memory base`	Shows the absolute address for processor exceptions. Also helpful for address allocation in shared flash memory regions.
`External bus width`	Indicates the width of the bus connecting external components. Automatically updated when components are connected to processor buses.
`External Memory span`	Displays the length of the external bus. Measured in units specified in the "External Memory Unit" field.
`External Memory unit`	Specifies the unit of measurement for the External Memory span.
`Device`	Shows the selected FPGA device for implementation. Crucial for understanding hardware constraints and capabilities.
`GEMRTOS Interrupts`	Indicates the number of interrupts configured in the GeMRTOS IP. Important for understanding the system's interrupt handling capacity.

This information tab serves as a quick reference for system designers and debuggers. It provides a snapshot of critical system parameters, helping to ensure that the configuration aligns with the intended design and assisting in troubleshooting efforts. The data presented here can be particularly useful when defining memory maps, calculating timing requirements, or verifying interrupt configurations.

By presenting this information in an easily accessible format, the GeMRTOS IP information tab streamlines the design process and helps prevent configuration errors that could arise from misunderstanding or overlooking key system parameters.

6.5 GeMRTOS IP Memory Map

The GeMRTOS IP handles memory mapping for all its internal components which are connected to the system bus. This section details the memory allocation process and the structure of the internal memory map.

Memory Map Assignment

- The GeMRTOS IP automatically assigns memory map addresses to all its internal components connected to the system bus.
- The processor with `CPU_ID` 1 determines the system's memory map (useful for BSP project generation).
- The memory addresses assigned by `CPU_ID` 1 are used by the compiler when generating the application.

Internal Memory Map Components

The internal memory map includes the following components:

Table 6.3: GeMRTOS IP internal memory map.

Component	Description
`processor debug memory`	Memory region dedicated to debug capabilities of the processor, if available.
`gemrtos_global_region`	Memory region of the global register of the GeMRTOS controller.
`gemrtos_processor_region`	Processor shadow memory region in the GeMRTOS controller.
`jtag_uart_0`	JTAG UART component for STDIO when enabled in GeMRTOS parameter window.
`jtag_uart_1`	JTAG UART component for STDERR when enabled in GeMRTOS parameter window.
`jtag_uart_2..Processors+1`	JTAG UART component for each processor when enabled in GeMRTOS parameter window.

6.5.1 Memory Map Organization

All memory-mapped components listed in Table 1.3 are placed in memory addresses after (higher than) the external components connected to the GeMRTOS IP component in the System Contents window.

This memory mapping scheme ensures:

- Each component has a dedicated address space.
- A consistent memory layout across the system.
- Proper integration of debug and communication features (e.g., JTAG UARTs).
- Correct memory location referencing by the compiler when generating code.

Understanding this memory map is crucial for developers working with GeMRTOS, as it impacts how they access various system components and debug their applications. It also highlights the system's flexibility in accommodating different configurations of processors and communication interfaces.

6.5.2 Conclusion

The GeMRTOS IP's memory mapping capabilities are a crucial component of its overall system architecture. By automatically assigning memory addresses to connected components, the GeMRTOS IP ensures a consistent and organized memory layout across the entire system.

The central role of the `CPU_ID` 1 processor in defining the memory map is particularly noteworthy. This design choice allows for a unified memory representation, which is essential for the compiler to correctly generate application code that can seamlessly interact with the various system resources. The memory addresses determined by `CPU_ID` 1 serve as the reference point for the entire software ecosystem, streamlining development and integration efforts.

The internal memory map components, such as the processor debug memory, GeMRTOS global and processor regions, and the JTAG UART interfaces, provide valuable system-level features and capabilities. By strategically placing these components in the memory hierarchy, the GeMRTOS IP ensures that they can be efficiently accessed and utilized by both hardware and software components, enhancing the overall system's functionality

and debugging capabilities.

Ultimately, the GeMRTOS IP's memory mapping approach demonstrates a well-designed and comprehensive strategy for managing the complex interplay of hardware and software within a real-time embedded system. This level of attention to memory organization and accessibility is a testament to the GeMRTOS IP's commitment to delivering a robust and versatile platform for developers working on demanding, time-critical applications.

CHAPTER 7

GeMRTOS Hardware and Software Design Flow

Note: This chapter provides a conceptual overview of the hardware and software design flow. For the complete step-by-step walkthrough — including Platform Designer connection steps, Quartus Prime compilation, BSP generation, and application build — see Chapter 8.

7.1 Introduction

GeMRTOS makes the process of designing an efficient multiprocessor system as easy as a uniprocessor one. The hardware and software design flow involves the same tools that are used when designing a uniprocessor system based on Nios II or Nios V soft-processors. This consistency in the design approach simplifies the transition from uniprocessor to multiprocessor architectures, allowing developers to leverage their existing knowledge and expertise.

The GeMRTOS IP provides a comprehensive solution that seamlessly integrates hardware and software components, enabling developers to focus on their application-specific requirements rather than the underlying complexity of a multiprocessor system. By abstracting away the low-level details, GeMRTOS empowers engineers to rapidly prototype, configure, and deploy their designs, ultimately accelerating the development lifecycle.

One of the key advantages of the GeMRTOS design flow is its ability to adapt to the specific needs of each project. Whether the system requires a single processor or multiple processors, the design process remains largely unchanged, ensuring a familiar and efficient workflow. This flexibility allows developers to scale their designs as the project requirements evolve, without the need for a complete architectural overhaul.

Furthermore, the GeMRTOS design flow leverages the well-established Nios II and Nios V development ecosystems. Developers can continue to utilize the same Quartus Prime and Altera SoC FPGA Embedded Development Suite tools they are already familiar with, minimizing the learning curve and enabling a seamless integration with existing development practices. This approach helps to preserve investments in existing toolchains and ensures a smooth transition to multiprocessor architectures.

7.2 Quartus Prime / Platform Designer Hardware Design Flow

Quartus Prime is a sophisticated software suite developed by Altera for designing, synthesizing, and verifying digital circuits targeting Altera FPGAs, SoCs, and complex programmable logic devices (CPLDs). It provides a comprehensive environment that enables designers to take a project from initial design entry to final programming of the target device.

Platform Designer, formerly known as Qsys, is a system integration tool within the Quartus Prime software. It automates the process of connecting different intellectual property (IP) components and subsystems within an FPGA or SoC design. Essentially, it handles the complex task of generating the interconnect logic needed to link these components, saving significant time and effort in the design process.

The GeMRTOS IP integrates seamlessly into Platform Designer, as it is available as any other IP component in Platform Designer. Once installed, the GeMRTOS Multiprocessor IP is easily accessible from the IP Catalog in Platform Designer, as shown in 2.1.

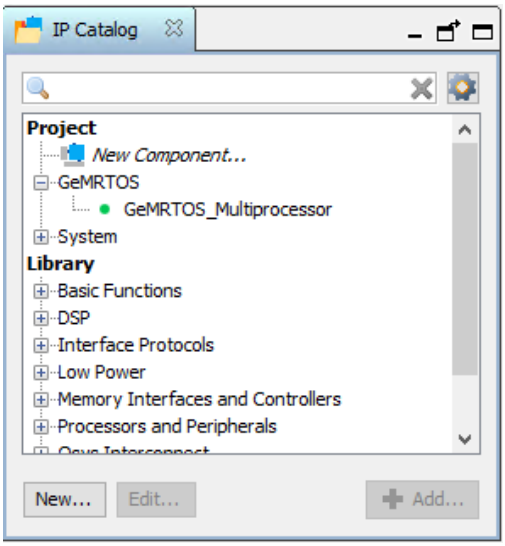

Figure 7.1: GeMRTOS IP integration in Platform Designer IP Catalog.

The GeMRTOS IP may be integrated into any project as if it were a Nios II or Nios V soft-processor by adding and connecting it in the System Contents window of Platform Designer.

Figure 2.2 shows the design flow of a project using Platform Designer and Quartus Prime.

The design flow starts by including the GeMRTOS Multiprocessor component into a Platform Designer project. A Platform Designer project is saved in a file with the extension `.qsys`. This Platform Designer project contains all the components and connections of the SOPC design.

When the SOPC is designed, it should be generated using the "Generate HDL" command in Platform Designer. The Platform Designer generation process produces the following files:

- `<sopc_system>.cmp`: a plain-text file including the declaration of the component that contains all the SOPC hardware project. This allows the SOPC to be instantiated into a Quartus Prime HDL project as a hardware component for further compilation to a targeted FPGA device. The `.cmp` file shows the input and output conduit ports defined in the Platform Designer SOPC for easy integration into the HDL project in Quartus Prime.

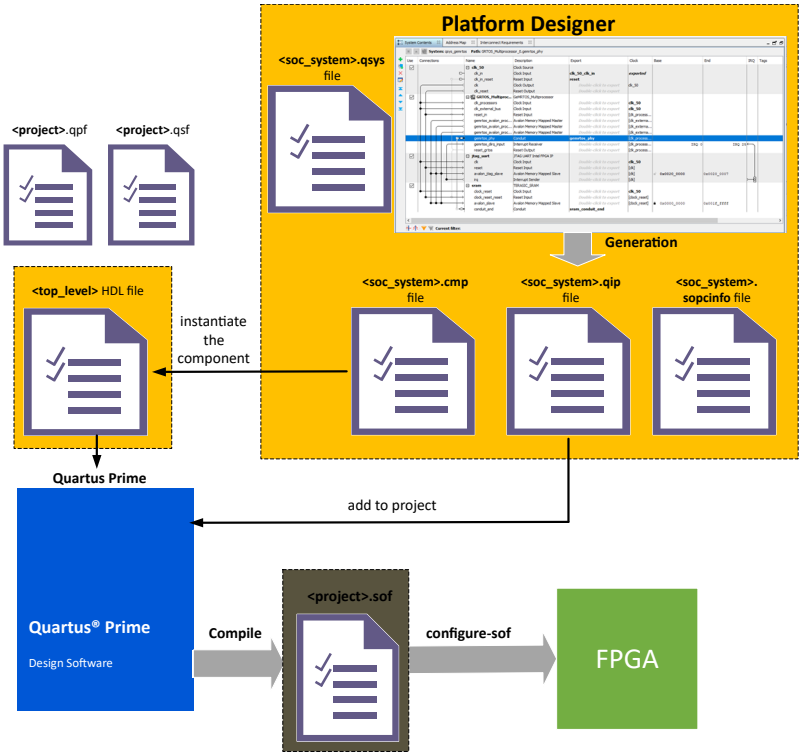

Figure 7.2: Platform Designer and Quartus Prime design flow.

- `<sopc_system>.qip`: a Quartus Prime IP project file that includes all the information about the SOPC project created in Platform Designer to be included in Quartus Prime for compilation. The `.qip` file should be included into the Quartus Prime project in order to make all the components included in the SOPC project available.
- `<sopc_system>.sopcinfo`: a file that contains all the SOPC information required to generate the software support for the SOPC system created with Platform Designer. System defines, memory map addresses, and system components are detailed in the `.sopcinfo` file for software interface generation.

As it can be seen, Platform Designer allows creating an SOPC project and also generates all the files and information required to produce both the hardware and the software components of the system.

Quartus Prime is then used to generate the FPGA configuration file from the hardware description, which should include the Platform Designer component that integrates the GeMRTOS IP for implementing the multiprocessor system. Quartus Prime is also utilized for programming the FPGA device with the configuration file produced.

7.3 The Software Design Flow

The software design flow involves several steps to create an application from the hardware specification, given in the `.sopcinfo` file, to the full compilation of the application code.

For Nios II and Nios V processors, the software creation involves two software projects:

- **BSP (Board Support Package) Project**: This project contains all the drivers and system definitions required to compile, initialize, and manage all the hardware components included in the SOPC project. For instance, when a JTAG UART component is included in the SOPC project, the initialization routines for that component are included in the system initialization, and the read and write functions are made available for the user to access the functionality of the JTAG UART device. All the information required to create the BSP project is included in the `.sopcinfo` file. Each time the Platform Designer project is modified, the BSP project should be regenerated to update its software components.
- **Application Project**: This is the software project that implements the functionality required by the application. It uses the BSP project interfaces to access the hardware devices of the system. For example, a `printf()` instruction may invoke the writing routines for the `STDOUT` JTAG UART of the system.

Figure 2.3 shows the different layers covered by the BSP and Application projects.

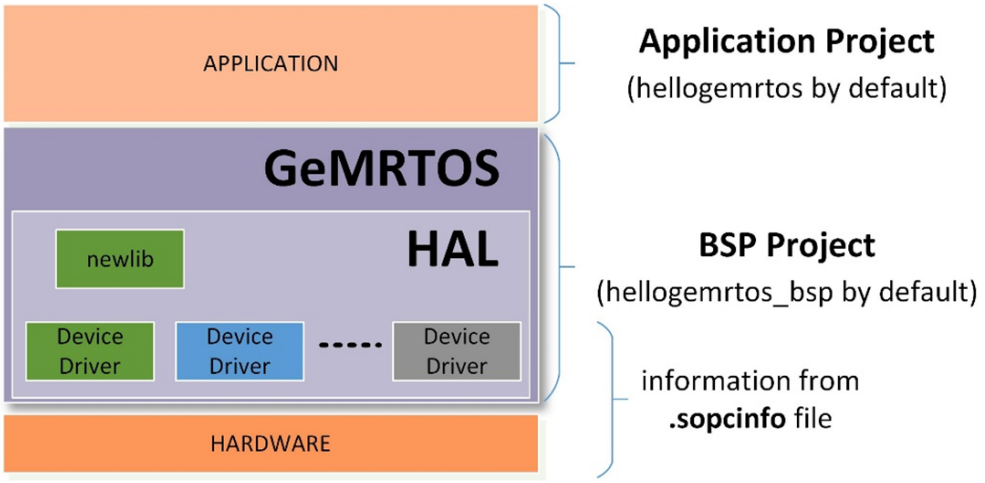

Figure 7.3: GeMRTOS software layer containing the BSP and Application projects.

The software design flow involves several steps that must be taken in the correct order.

Note: The Nios II Command Shell requires a Linux-like environment. Earlier versions of Quartus Prime utilized the Cygwin environment to execute these commands. However, Quartus Prime versions later than 22.0 now use the Windows Subsystem for Linux (WSL) command-line environment. The procedures for installing Quartus Prime version 22.0 and configuring WSL for the GeMRTOS design flow on Windows 10-64bit systems are detailed in gemrtos.com.

By following the established software design flow and utilizing the BSP and Application projects, developers can effectively leverage the GeMRTOS IP and create robust, multi-threaded applications that take full advantage of the underlying hardware architecture.

7.3.1 The `settings.bsp` file

The `settings.bsp` file is the file that contains all the configuration of the BSP project. Memory regions, device drivers, parameter settings, and the main processor are included in the `settings.bsp` file for use in compilation.

The `niosv-bsp` utility is used to create and regenerate the `settings.bsp` file. The `niosv-bsp-editor` utility

(Figure 2.4) may be used for graphical editing of the settings in the `settings.bsp` file.

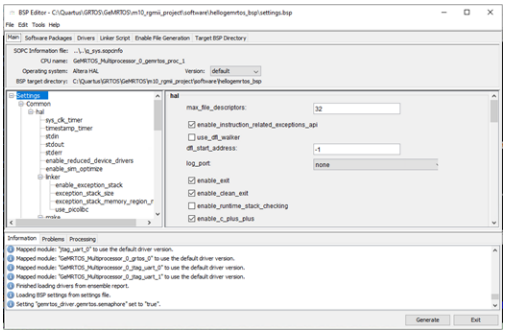

Figure 7.4: The `niosv-bsp-editor` utility for editing the `settings.bsp` file.

GeMRTOS requires some HAL (Hardware Abstraction Layer) settings for better performance that are set by default when implementing the software design flow through the `gemrtos_build` script:

- `hal.sys_clk_timer` (`none`): Usually, most SOPC designs define a timer component for time support in HAL. However, this feature is not required for GeMRTOS, and consequently, it should be set to `none`.
- `hal.timestamp_timer` (`none`): The timestamp timer may be set when debugging features are required. In GeMRTOS, this feature is not required, and consequently, it should be set to `none`.
- `hal.enable_reduced_device_drivers` (`true`): By setting `enable_reduced_device_drivers` to `true`, the device drivers avoid interrupt alternatives. In GeMRTOS, it should be convenient to avoid using HAL interrupts instead of GeMRTOS triggers.
- `enable_instruction_related_exceptions` (`true`): This setting allows catching the processor exceptions to avoid runtime errors.
- `hal.stdin`, `hal.stdout`, and `hal.stderr`: These settings define the JTAG UART devices for `STDIN`, `STDOUT`, and `STDERR` consoles, respectively. In GeMRTOS, by default, these are set to `jtag_uart_0`, `jtag_uart_0`, and `jtag_uart_1` for `STDIN`, `STDOUT`, and `STDERR`, respectively.

GeMRTOS generates the `gemrtos_build` script for automated software development, which is discussed in the next section.

7.4 `gemrtos_build` Script for Software Design Flow

The software design flow involves several steps for configuring the different projects involved. Figure 2.5 shows the software design flow detailing the main files involved.

The `gemrtos_build` script executes several commands to process the BSP and application projects to produce the `<application>.elf` file that contains the configuration file ready to download to the GeMRTOS multiprocessor system.

The main steps performed by the `gemrtos_build` script are:

- **Generate the BSP Project**: The script generates the BSP project based on the configuration specified in the `.sopcinfo` file and the settings in the `settings.bsp` file. This creates the necessary device drivers, system definitions, and other components required for the hardware platform.
- **Compile the BSP Project**: The script compiles the BSP project to produce the BSP library that can be used by the application project.
- **Compile the Application Project**: The script compiles the application project, which uses the BSP library to access the hardware devices and implement the desired functionality.

- **Link the Application Project**: The script links the compiled application code with the BSP library to produce the final `<application>.elf` file.

By running the `gemrtos_build` script, developers can streamline the software design flow and ensure that the necessary steps are performed correctly to produce the final executable file that can be downloaded to the GeMRTOS multiprocessor system.

The `gemrtos_build` script abstracts away the complex details of the software design flow, allowing developers to focus on the application-level development while ensuring that the underlying hardware-software integration is properly handled.

7.5 Conclusions

This chapter has provided a comprehensive overview of the hardware and software design flow for GeMRTOS-based multiprocessor systems. It highlighted the seamless integration of GeMRTOS IP into Altera's Platform Designer, allowing developers to leverage familiar tools and methodologies for efficient system creation. The streamlined design flow, encompassing both hardware configuration in Quartus Prime and software development using the Nios II or Nios V toolchain, ensures a cohesive and predictable development experience.

A key aspect of this flow is the automated generation of the Board Support Package (BSP), which contains the drivers and system definitions necessary for software development. The `gemrtos_build` script simplifies the software build process, automating tasks such as BSP configuration, application compilation, and FPGA programming. This reduces development time and minimizes the potential for manual configuration errors.

Finally, by adhering to a well-defined and repeatable design flow, developers can harness the full potential of GeMRTOS to create innovative and adaptable multiprocessor systems. The automation of key steps, combined with the availability of comprehensive software libraries and reference designs, empowers developers to confidently tackle complex embedded projects and deliver high-quality solutions.

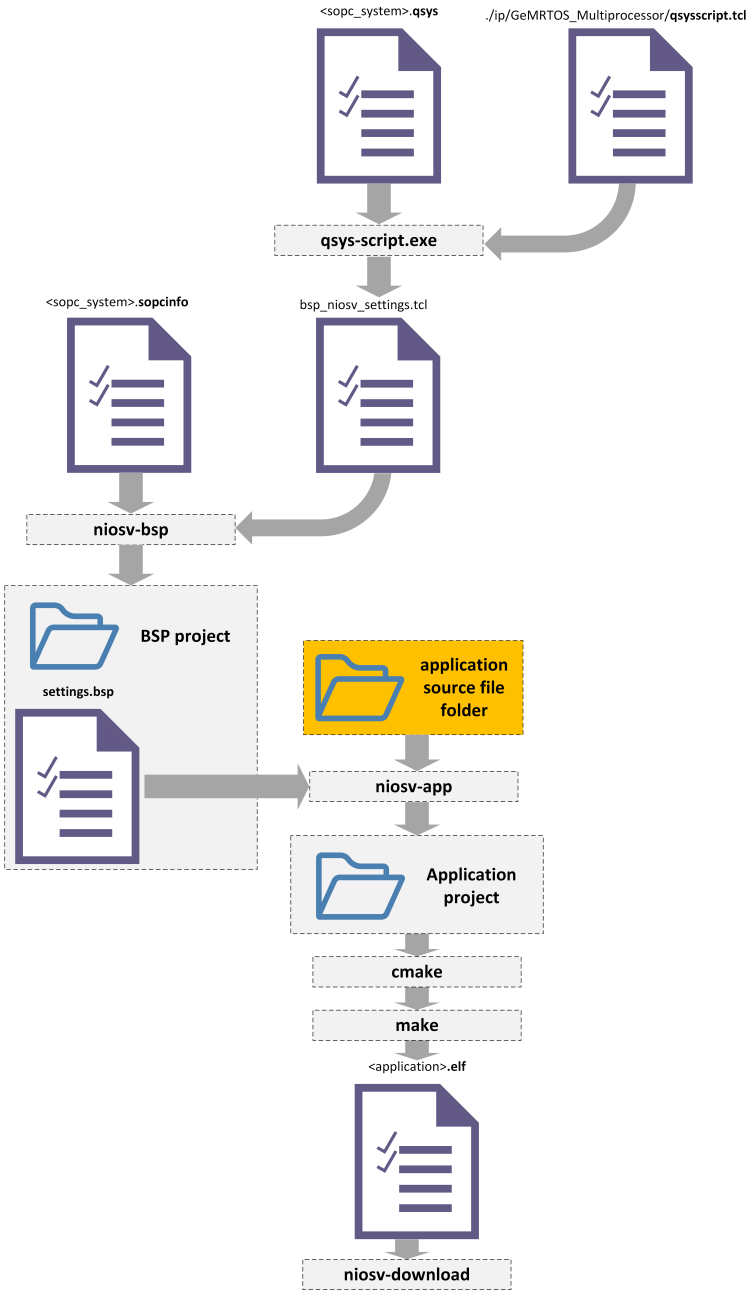

Figure 7.5: The `gemrtos_build` script software flow.