SiFive WorldGuard Technical Paper

SiFive Shield Open Secure Platform Architecture

Version 2.1

SiFive WorldGuard Technical Paper

Proprietary Notice

Release Information

Version	Date	Changes
2.1	July 14, 2021	• Editorial fixes
2.0	April 16, 2021	• Updated from WorldGuard High-Level Specification v0.9
1.2	December 2, 2020	• Updates to wording, diagrams
1.1	September 25, 2020	• Updates to format, wording
1.0	July 27, 2020	• Initial release

Glossary

Term	Meaning
AMO	Atomic Memory Operation
BEU	Bus Error Unit
CMO	Cache Maintenance Operation
CSR	Control and Status Register

Term	Meaning
FSBL	First Stage Boot Loader
Initiator	The inclusive word for "master" block
MMU	Memory Management Unit
PMC	Power Management Controller
Privilege modes	*The RISC-V Instruction Set Manual, Volume II: Privileged Architecture* defines up to five privilege modes: M, [H]S, U, VS, VU
REE	Rich Execution Environment
SBI	Supervisor Binary Interface
Target	The inclusive word for "slave" block
TEE	Trusted Execution Environment
WID	WorldGuard World Identifier
WG	WorldGuard
`XLEN`	Refers to the width of an integer register in bits (either 32 or 64)

Contents

1

Summary

SiFive® Shield is an Open Secure Platform Architecture that includes a secure boot, cryptographic engines, and a hardware-enforced multi-domain solution named SiFive WorldGuard. WorldGuard is a hardware-based software isolation solution for SiFive cores and applies to physical addresses. For the rest of the document, "wg" may be used to denote WorldGuard.

2

Introduction

Software isolation is an important feature mandated by distinct concerns. The trend of increasing code size, different origins of code, and the ways they are combined can introduce important risks such that one part can impact another, intentionally or not.

If there is a bug that causes unintentional malfunction of the SoC software, then there is a concern about safety. A health-oriented device (such as a medical device, car brake, nuclear plant sensor) must maintain its main mission, even if another piece of software running on the device has bugs, goes into an infinite loop, or displays erratic behavior. If there is a weakness that allows an intentional (nefarious) act to bug the software, then there's a concern about security. A credential-oriented device (such as a payment terminal, badge control system, DRM-equipped device) must resist or raise a flag and not compromise any asset it protects. Then, it makes sense to develop solutions that guarantee that the failure of one piece of software does not impact the correct and full functioning of other pieces running on the same platform.

Beyond that, it makes sense to guarantee that an aggressive or uncontrolled piece of software cannot impact a resource belonging to another piece of software running on the same platform. Impact on a resource can mean a broad number of things, including simply reading a portion of memory, modifying this portion of memory, or preventing the use of a peripheral. As such, it is important to note that security requirements are more demanding than safety requirements. The latter usually leads to solutions where the application code is able to change its own privilege mode, while the former usually requires that the application code cannot change privilege modes. It is then obvious that safety schemes, usually based on a regular real-time OS, cannot comply with security requirements, while security schemes can generally comply with safety requirements.

WorldGuard is a hardware-enforced multi-domain solution that provides protection against illegal memory or peripheral accesses from software applications and other initiators (formerly known as "masters"), like DMAs. It provides hardware with the means to guarantee strict and strong isolation between distinct software applications running on the same single or multi-core platform and accessing shared resources, such as memories and peripherals.

This solution is intended for all SiFive Core IP products and any software architecture. It is fully compatible with and does not require any modification to the RISC-V ISA.

The SiFive Shield solution proposes to create "worlds" that gather applications running on cores and other initiators, like DMA channels, and resources like memories or peripherals. A world defines permissions on physical addresses in the system, can overlap, and does not affect software-visible cache coherence. Software isolation is achieved by restricting a world’s access to specific physical addresses that would be bound to different components in a system. A world in a WorldGuard-enabled system or chip will be identified by a WorldGuard Identifier or WID.

The possible number of worlds is hardware-configurable at elaboration stage and should be determined by the number of cores and initiators, the memories' configurations, and, most importantly, the software architecture.

This solution does not replace the RISC-V core PMP mechanism (applicable to a single core for memories), nor does it replace the Memory Management Unit (MMU) but extends them to a multi-core system with other initiators and provides stronger security.

There is, however, a major limitation to the PMP and privilege modes approach for more complex multi-core platforms. As the privilege modes and the PMP scopes are restricted to their core and their software, they’re only able to control the memory-mapped areas of a single core. The privilege management registers belong to that single core, so their values and the resulting controls are neither shared nor "agreed" with other cores, thus mandating a complex synchronization effort. In that case, the usage of a *trusted core* may be preferred.

Another limitation of the PMP approach applies to platforms with a "fractured" physical memory address map. In this context, the number of required PMP entries may become too large to be acceptable. The SiFive WorldGuard solution is a good alternative.

3

Features

- Intended for all SiFive cores
- Controls physical addresses
- Provides access control on resources (memories, peripherals) in read, write, and execute (optional, if available)
- The maximum supported number of worlds is set at the elaboration stage
- World *trusted WID* "golden value" provides configuration privileges
- World *null WID* provides no access at all, to any resource
- Modifies L1 and L2 cache tags for WID support
- Supports two kinds of cores: "wg-aware" cores that can tune their WID value, and non-wg-aware cores that are not aware of the WorldGuard solution at all
- Privilege mode WID CSR in wg-aware cores is managed by upper privilege modes
- Supports additional CSRs in wg-aware cores
- In wg-aware cores, M-mode local configuration privileges (including WID changing privilege) can be delegated to local lower privilege modes
- Error interrupts are trapped in M-mode
- Memory regions in wgChecker have sizes multiple of 4KB
- Worlds' resources may overlap
- Non-wg-aware cores do not have their WID depend on the privilege mode
- WG-aware cores can have their WID depend on the privilege mode
- On wg-aware cores, M-mode local configuration privileges (including WID changing privilege) can be delegated to local lower privilege modes
- Provides two main blocks: the wgMarkers and the wgCheckers (with different flavors)
- Provides the concept of a *trusted agent*, being either a dedicated core or M-mode firmware
- World-based debug mechanisms

- `XLEN` is 32 or 64 bits

The wgMarkers add a WID to bus transactions passing through them; they have in and out ports, and wires or a control port for internal register configuration. A wgMarker cannot tag a bus transaction that is already tagged with a WID. The wgMarker can also be restricted to only tag a transaction with a subset of allowed worlds.

The wgCheckers accept or reject transactions based on WID, and capture information on failed transactions in local registers. On rejecting a transaction, loads and instruction fetches would typically return zero and stores are acknowledged, but not sent further downstream. Whether a failed transaction generates an interrupt using an access denied response is configurable and is discussed later in this document.

A *trusted core* is a core that can generate transactions in the most trusted world. It is a core that owns the highest WorldGuard privilege level, thus belonging to the *trusted world* and holds the *trusted WID*. It is unlikely that a non-wg-aware core is a *trusted core*.

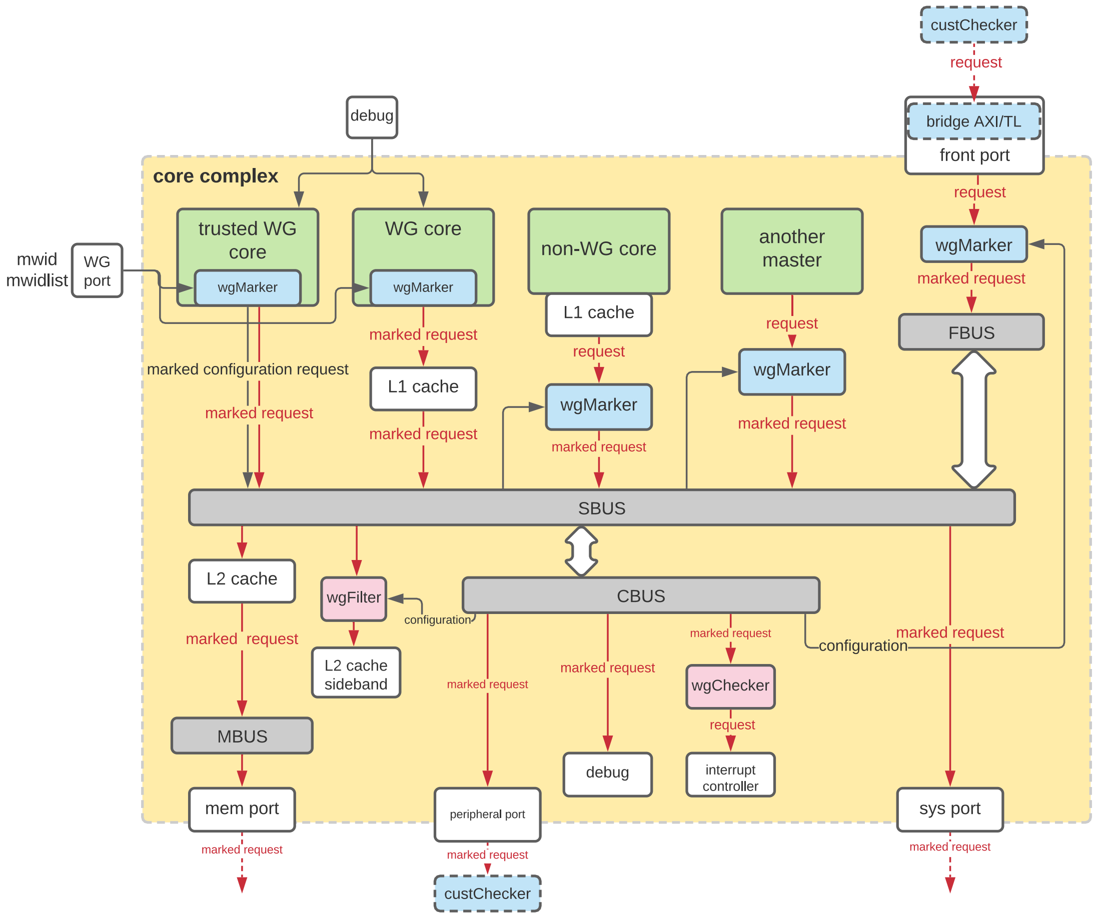

Figure 1: *WorldGuard Generic Block Diagram*

The block diagram above describes the following:

- Only the WorldGuard solution and blocks within the Core Complex are described
    - What is performed outside the Core Complex is user-dependent, especially the Checkers. The same principles used for wgCheckers apply for user Checkers (custChecker).
- The initiators (in green) are all equipped with a wgMarker that mark their outgoing transactions
    - The wg-aware cores have their wgMarker within the tile, before the L1 cache (the L1 cache lines contain WID-related information)
    - The non-wg-aware cores and other initiators have their wgMarker outside the core, after the L1 cache (the L1 cache lines do not contain any WID-related information)
- The *trusted agent*, identified here as a *trusted core*, configures the wgMarkers (in blue) and the wgCheckers (in pink) by sending transactions marked with the *trusted WID*. In the absence of a trusted core, M-mode firmware in one or more of the wg-aware cores will act as the trusted agent.
- A Front Port located at the entrance of the Core Complex may receive transactions pre-filtered outside of the Core Complex by custChecker
- The transactions entering via the Front Port are marked
- Any marked transaction goes through the wg-aware bus and blocks (e.g., L2 cache) until the targeted wgChecker is reached
    - A wg-aware L2 cache has its cache lines containing the WID as part of the tag
- The wgChecker (regardless of complexity) determines if the WID and related meta-data (access type) are consistent with the rules it contains; if it matches, the transaction is authorized
    - Simple wgCheckers are used for controlling marked configuration transactions in wgMarkers and wgCheckers for their inner settings setup
    - More complex wgCheckers are used for controlling marked transactions accessing simple resources such as peripherals or interrupts controllers (in a go/no-go strategy)
    - Ultimately, complex wgCheckers are used for controlling complex resources such as memories, in a way slightly equivalent to what RISC-V core PMPs do
- The transaction, freed from its WID information, is sent to either the resource (interrupts controller, debug module) or the outgoing port (Memory Port, Peripheral Port, System Port)

Note

In the block diagram, a custChecker is present after the Peripheral Port, outside of the Core Complex. Alternatively, a wgChecker can be placed before the port, within the Core Complex. Checkers can be placed either inside or outside of the Core Complex. If the latter is applied, the bridge within the port must handle the translation of the WID field into another means of tagging the transaction.

4

Security Rationale

The WorldGuard solution offers a system-level approach to securing access to system resources by software applications.

- At the system level, the highest trust is considered to be limited to the following hardware: the secure boot code in ROM and the *trusted agent*
- The *trusted agent* can later on include other cores in the *trusted perimeter*; it may also delegate some local rights to wg-aware cores
- The WorldGuard definitions and gate-keeping mechanisms prevent any initiator, including cores and the code running on them, from gaining unauthorized privileges at the system level
    - PMP configuration cannot override the WorldGuard configuration, therefore, any code running on a core cannot gain more privilege than what is limited to its world
    - MMU management involves a large piece of code that is not acceptable for a satisfying level of trust; conversely, the isolation provided by the WorldGuard mechanism requires a very small portion of software, acceptable in a Trusted Code Base (TCB)

SiFive WorldGuard is a software isolation solution for an aggressive environment by considering situations where entities exploit software vulnerabilities to take control of the whole system. Typically, these exploits aim at gaining privileged access to restricted resources. The principle of protection enforced by WorldGuard is to circumscribe each of these, potentially aggressive and/or under attack, entities within a world.

For the current SiFive WorldGuard solution, we only consider software and non-invasive attacks.

5

Use Cases

The WorldGuard isolation solution proposes on one side to "mark" the requests issued by the initiators and sent to the resources through the communication bus, and on the other side to filter those marked requests at the resources' side, based on a set of filtering rules. The process is to define worlds made of initiators (e.g., one or several cores, a DMA channel) and resources (e.g., some portions of memories, some peripherals). These worlds are uniquely identified by World Identifiers, a.k.a. WID. Once the worlds' definition is completed, the corresponding configuration process on the platform must be initiated and controlled by an authority with the highest privileges. This authority belongs to a specific world, named the *trusted world*, associated with the *trusted WID*. Beyond this specific world holding the platform-level highest privileges, the application software and operating systems, assigned with lower privileges, may receive delegations for some local WorldGuard settings on wg-aware cores.

Depending on the platform architecture, this authority, named the *trusted agent*, may have a different form; it can be a dedicated core, or multiple cores that can hold and share this privilege, while also running application software. Some use cases are presented here to look at possible variations.

Except for the variety in *trusted agents*, the whole system uses the same blocks: wgMarkers for marking the requests, and the wgCheckers for checking the marked requests (these blocks are detailed in Section 6.2). Only the *trusted WID* value grants access to non-wg-aware cores and blocks configuration, all other WIDs are rejected. The wg-aware cores can also be locally configured (to a certain extent) using CSRs.

5.1 Multi-core SoC with Trusted Core

A high level of security can be reached on a multi-core platform by using a distinct, dedicated core; this so-called *trusted core*, and associated firmware, represents the *trusted agent*.

This *trusted agent* is in charge of the WorldGuard configuration, and so the only one to belong to the *trusted world* and flagged with the *trusted WID*. The other cores and their software are not able to configure and/or access WorldGuard blocks. This kind of platform is usually complex and hosts several initiators, like DMA, cryptographic blocks, many peripherals and memories and is identified as requiring a very high level of security. On some platforms the dedicated core may be in charge of other tasks such security-related tasks (cryptography, keys management) and power-management-related tasks (such as behaving as a PMC).

In this use case, the *trusted WID* must not be shared with an application core, which would disallow firmware running on this application core to configure WorldGuard blocks. The World-Guard trust must only be located at the *trusted core*. This prevents synchronization concerns and system-wide spread of attacks.

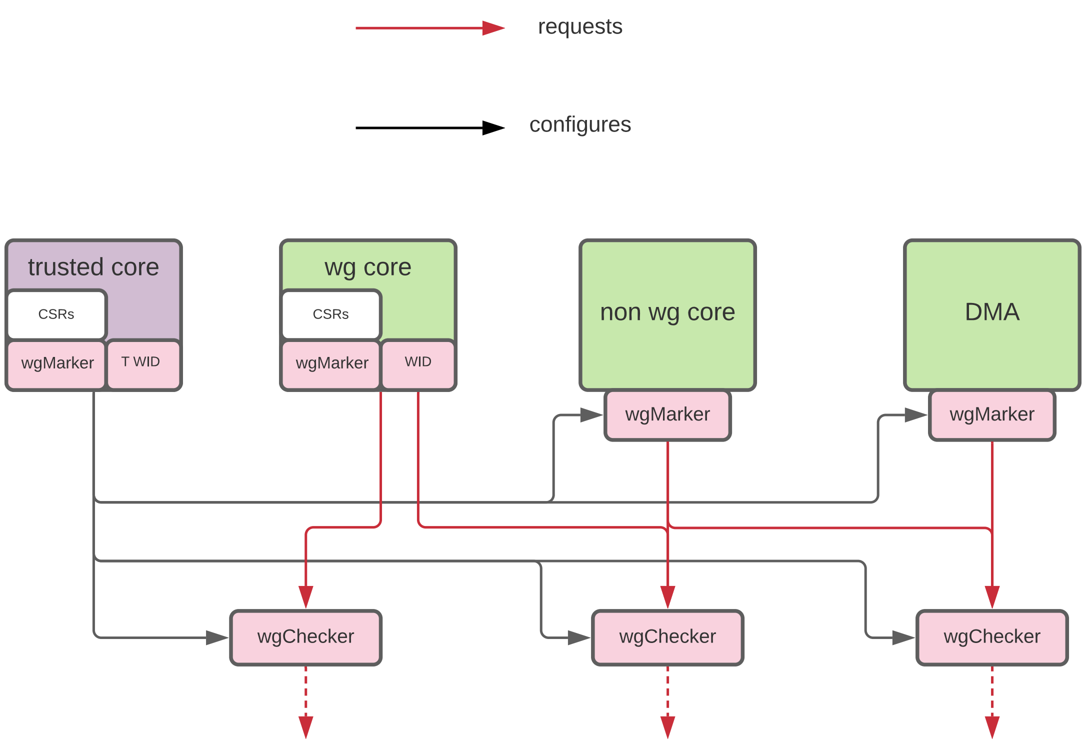

Figure 2: *Multi-core SoC with Trusted Core*

However, in this scenario, the *trusted agent* may delegate some privilege rights to any of the wg-aware cores' M-mode firmware. For example, this firmware can have the ability to modify its own core's wgMarker WID value, without the need to refer to the *trusted agent*. Obviously, this changed value has to be authorized in a certain range, this range being set by the *trusted agent* at the initial configuration stage. Locally to the core, the same scenario can apply and the M-mode firmware, which can also delegate some rights to the lower modes (if any). These local configurations are performed through dedicated CSRs.

For a multi-core system with both wg-aware and non-wg-aware cores, the *trusted core* would control the wgMarkers for transactions exiting the non-wg-aware cores.

5.2 Multi-core with M-mode Firmware as Trusted Agent

On a multi-core platform where the M-mode firmware on (some of) the cores is considered the *trusted agent(s)*.

One (or several) wg-aware core(s) have been granted the right to use the *trusted WID* by the initial configuration setup. This privilege allows the core(s) M-mode firmware to change the non-wg-aware initiators' wgMarkers' WID values and wgCheckers' configurations. The core-local wgMarker can still be accessed via the dedicated CSRs.

If multiple cores are granted this privilege, the M-mode firmware on any of these cores has the same level of privilege and the same level of trust for the whole platform. This also means that specific effort is required to synchronize the WorldGuard configuration changes all over the platform among the different M-mode firmware.

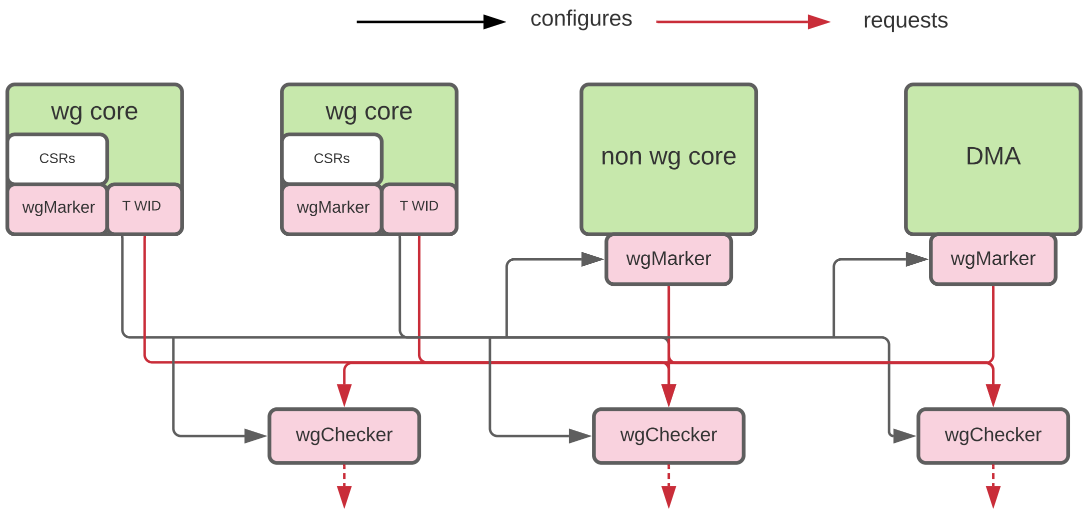

Figure 3: *Multi-core with M-mode Firmware as Trusted Agent*

This use case corresponds to a Linux server scenario. The platform hosts M-mode firmware in charge of the WorldGuard blocks configuration and runs the SBI firmware, interfaced with the S-mode Linux as emphasized in Figure 4. The WorldGuard configuration does not intervene within neither the Linux kernel nor the Linux userland tasks. There are two worlds operating in the software stack, one world for the M-mode firmware and one world for the upper layers, including S-mode and U-mode (both directly managed by the Linux kernel). In this case, the isolation between the tasks is managed by the kernel using an MMU, without any change from a vanilla Linux kernel. The WorldGuard solution transparently interoperates with the OS function and there is no configuration of any WorldGuard blocks by the Linux kernel.

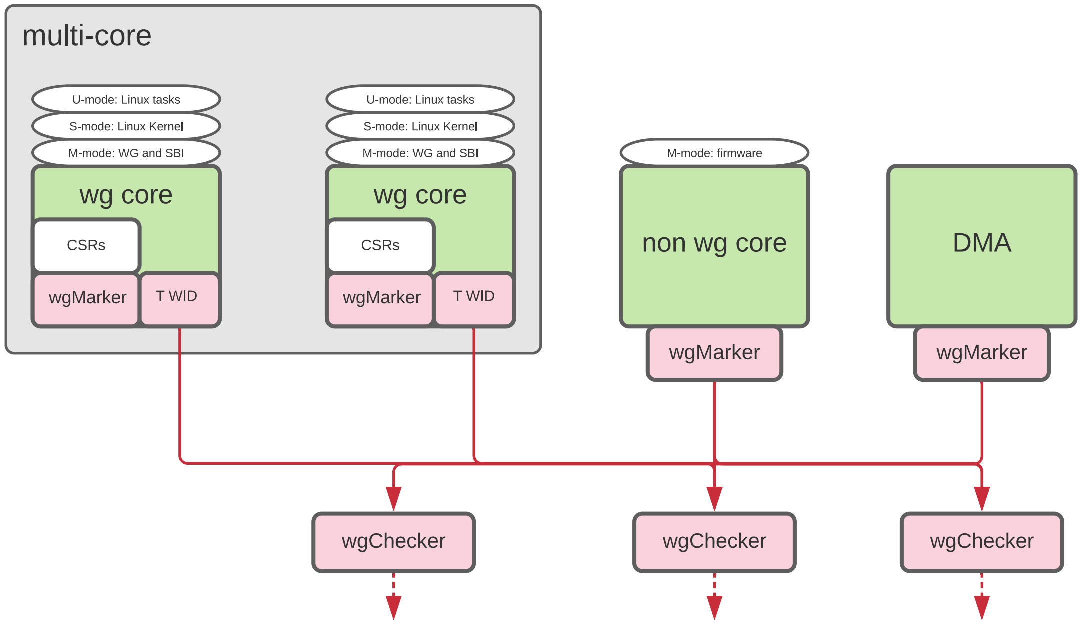

Figure 4: *Platform Running Linux on WG-aware Cores*

A derived use case could be a platform supporting the hypervisor privilege mode (HS-mode). In this case, a type-1 hypervisor firmware can operate on the top of the M-mode security monitor. Each container managed by the hypervisor runs in a distinct world; this container accesses the S- and U-mode only and is not aware of the WorldGuard solution (no configuration of any WorldGuard blocks by any container software). Figure 5 describes this architecture.

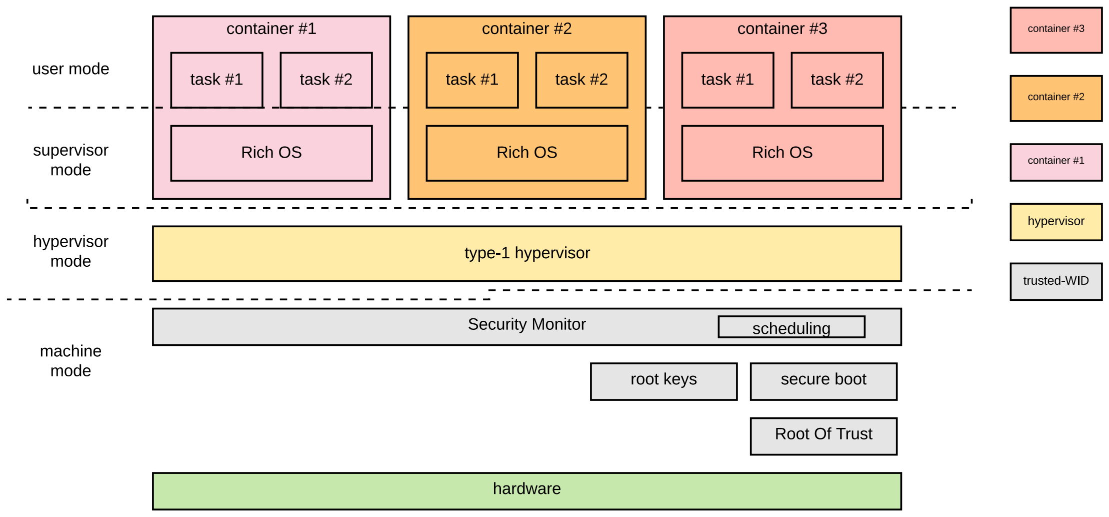

Figure 5: *Hypervisor Architecture with WorldGuard*

5.3 Single Core

On a platform with a single core, the M-mode firmware is considered the *trusted agent*.

The M-mode firmware is able to access the WorldGuard configuration and freely configure its own wgMarker WID via the core CSRs. Therefore, it can assign its own wgMarker the *trusted WID* value. This ability allows this firmware to change the other initiators' wgMarkers WID values and the wgCheckers' configurations.

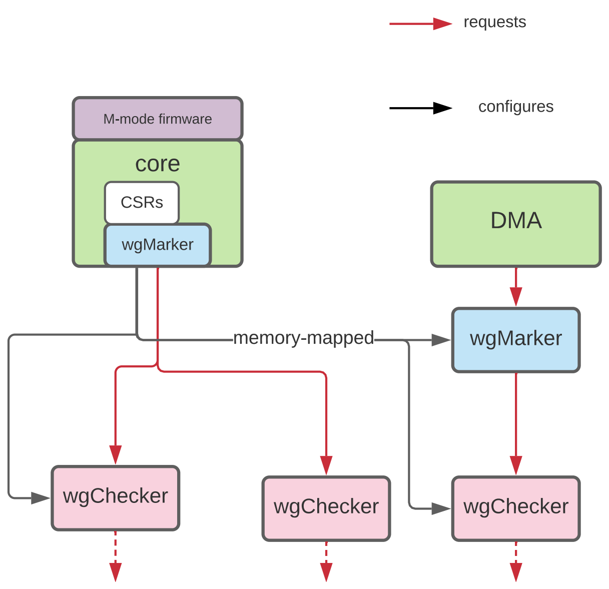

Figure 6: *Single-core with M-mode Firmware as Trusted Agent*

The first configuration in the single core use case is the RTOS scenario. In this scenario (see Figure 7), an RTOS running in either M-mode or (ideally) S-mode, manages different tasks in U-mode, and uses the WorldGuard mechanisms for improved task isolation. Ultimately, any U-mode task belongs to a different world. It prevents illegal access to resources. Even though the core PMP can solely play this role, some implementations find benefit in using the WorldGuard mechanism; a large fragmentation of protected resources can consume or require too many PMP entries while the WorldGuard worlds configuration is scalable.

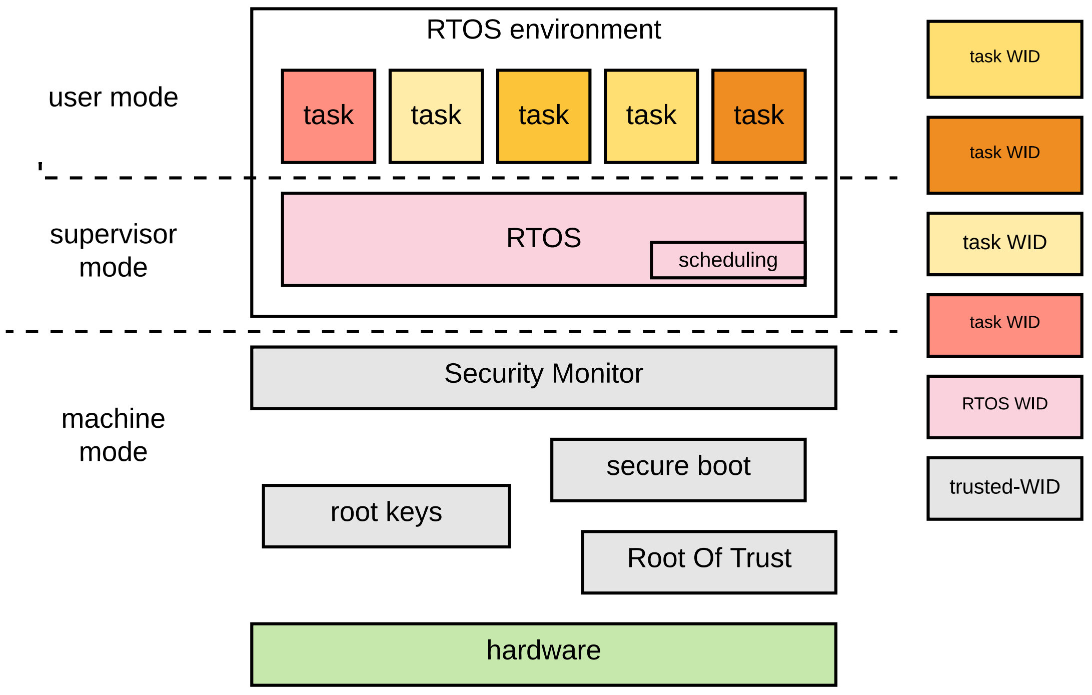

Figure 7: *RTOS Architecture with WorldGuard*

This single-core use case fits well with the REE/TEE software architecture, shown in Figure 8.

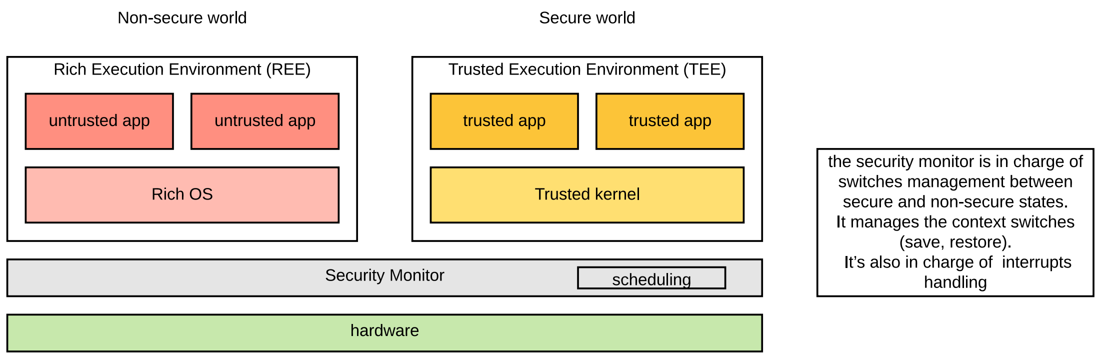
Non-secure world
Secure world
Rich Execution Environment (REE)
untrusted app
untrusted app
Rich OS
Trusted Execution Environment (TEE)
trusted app
trusted app
Trusted kernel
Security Monitor
scheduling
hardware
the security monitor is in charge of switches management between secure and non-secure states.
It manages the context switches (save, restore).
It's also in charge of interrupts handling

Figure 8: *REE/TEE Software Architecture*

This architecture defines two worlds, one being trusted and secure and the other being insecure and untrusted, or at least not trusted enough to access assets handled by the trusted world. Furthermore, the secure world, a.k.a. Trusted Execution Environment (TEE), contains different applications that are supposed to be isolated from each other. This architecture relies on security monitor firmware that is in charge of the secure switch between secure and non-secure worlds. The application of WorldGuard in this architecture leads to distinct worlds for the Rich Execution Environment (REE) and the TEE. It can even allow trusted applications to evolve in different worlds, providing a stronger isolation than usual, e.g., proposed by an MMU.

Figure 9 illustrates how a WorldGuard platform can address a REE/TEE architecture. The switch between different worlds in supervisor mode is managed by the security monitor while the switch between different user-mode apps can be delegated to either the Rich OS or the trusted kernel, respectively.

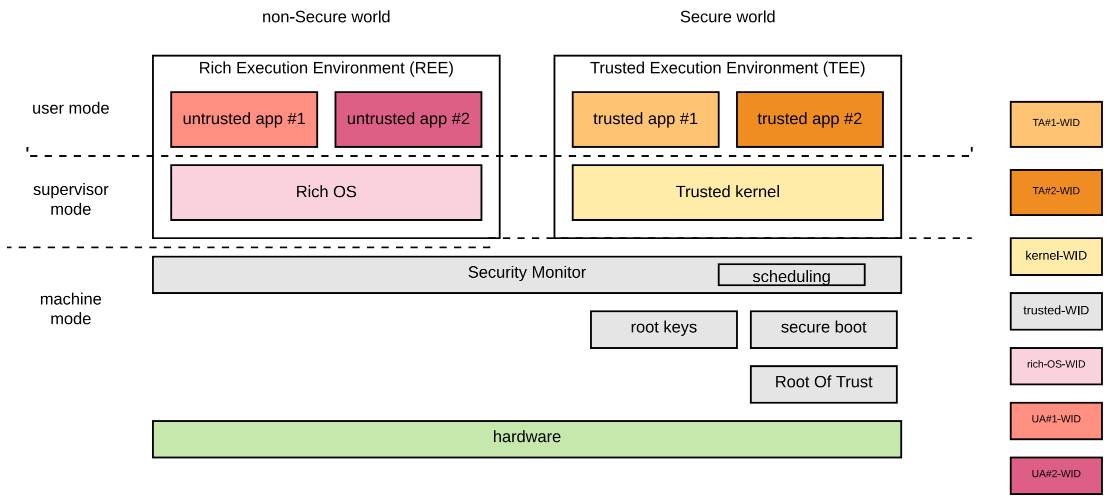

Figure 9: *REE/TEE Architecture with WorldGuard*

Note that this figure shows different worlds for even the untrusted user-mode applications (i.e., tasks). This scenario only applies if the Rich OS supports WorldGuard integration; this is something easily achievable with a RTOS (e.g., freeRTOS), however, is highly complicated in terms of the effort required to modify the codebase for a large OS (e.g., Linux). The availability of the H is not required for this architecture.

Note

This REE/TEE architecture applies to multi-core platforms as well and it is only described in the single-core use case for simplicity.

Beyond this secure/non-secure dual architecture, an example of a multi-level security architecture is the Keystone Enclave Architecture (see [3]), as described in Figure 10. This architecture considers on one side an untrusted world, typically a Linux or any other rich OS-based software container and one or more separate secure enclaves. In terms of versatility, the WorldGuard solution fits in will with the Keystone objectives.

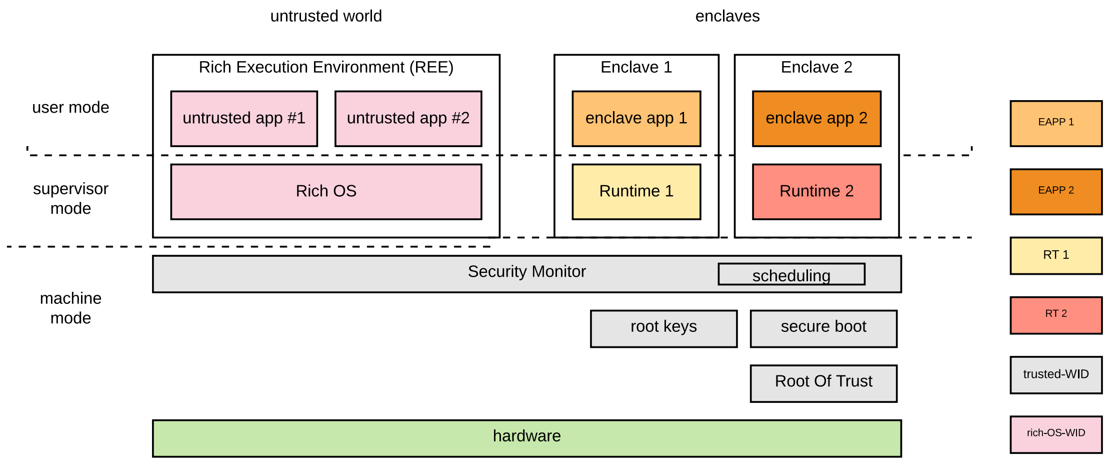

Figure 10: *Keystone Architecture with WorldGuard*

5.4 Bypass

On a platform equipped with WorldGuard, it may be the case that the software/product architect does not want to use the isolation mechanisms. The bypass use case would be to have a simple static configuration where only one world is defined, gathering all initiators and all resources. Therefore, any marked request is positively filtered (that is, accepted) by the wgCheckers and granted access, as in a non-WorldGuard platform. Consequently, there cannot be any illegal access detected by WorldGuard, and only software or other mechanisms can detect any violation.

It is recommended to lock this configuration once completed. The platform boots up with the M-mode WID coming via the wires on reset (described later in this document), so all accesses are open to M-mode. In bypass mode, all privilege modes (U, S, M) will use the M-mode WID, and thus, their accesses will not be controlled by WorldGuard.

6

Architecture

The principle of the WorldGuard architecture is that requests are marked at the initiator side and checked at the target (formerly known as "slave") side. The value the requests are marked with is determined by the wgMarker WID register. The way the marked request is checked is determined by wgChecker settings.

The ways wgMarkers and wgCheckers are configured are dedicated configurations accessible only by requests marked with the *trusted WID*. Core-local wgMarkers can be also configured by firmware at the M-mode privilege level.

6.1 WID Representation

The overall number of worlds is `NWorlds`, ranging from 0 to `NWorlds`. Two values in this set are reserved for specific purposes:

- The *trusted WID* value, equal to `Nworlds-1`, is used for granting access to the WorldGuard blocks' configuration registers
- The *null WID* value, equal to 0, is used for disabling any access, either for configuration or for usage

There are two kinds of WID representations:

- The real numeric value, code in binary format, requiring then $\log_2$`NWorlds`, for unique values selection
- The bit-mask value, requiring `NWorlds` bits, for multiple values selections. If `NWorlds` is less than the register length (i.e., `XLEN`), the remaining bits are RFU and set to 0.

The representation choice depends on the usage, e.g., a property shared by different worlds or an exclusive property. The registers' implementation describes the different choices.

6.2 Hardware Blocks

The hardware blocks described in this section are located in the Core Complex. Hardware blocks with similar features can also be located out of the Core Complex, but as they may be impacted by custom constraints and requirements, they won’t be described here. This section describes the blocks to be designed or modified:

- Adding CSRs in the wg-aware cores for managing the core WID values
- The initiator wgMarker manages the WID value that marks the requests coming out of an initiator
- The wgCheckers manage access rules to a resource (Memory Port, System Port, Peripheral Port) for a marked request
    - WGCheckers are wgBlockers, wgFilters and wgPMPs
    - The wgChecker (except wgBlockers) stores information about any violation

The CSRs and the core wgMarker are integrated in wg-aware cores. The initiator wgMarkers and the wgCheckers are memory-mapped and their control registers' access is blocked by the *trusted WID* value. None of the other WID values can modify any of these registers.

6.2.1 WorldGuard Port

A specific Core Complex Port (WG Port) is provided to connect the core WID value for M-mode and the `NWorlds` values for the wg-aware cores.

The port signals must be connected by the user at the integration stage of the Core Complex. These port signals are sampled to register at each Core Complex reset.

6.2.2 WGMarkers

A wgMarker adds a WID to bus transactions passing through. It has in and out ports, plus wires or a control port to configure.

A wgMarker cannot be used if input transactions already have a WID attached. This is a Chisel-level requirement, but it may also appear in any user-facing specification (in case they want to build wgMarkers).

A wgMarker can be restricted to only mark transactions with a subset of allowed worlds. The control port may allow more than the *trusted WID* to configure; this is the case for the wg-aware core CSRs.

The wgMarker is attached between an initiator and a bus, and it tags all bus transactions from this initiator.

Within the Core Complex, the wgMarker uses bits in the bus fields to carry the WID.

Two different categories of initiators can be identified:

- Cores that are wg-aware, allowing deeper integration of WorldGuard mechanisms within the core
- The other initiators, including non-wg-aware cores

6.2.3 WGMarkers for Non-wg-aware Initiators

The wgMarker is attached between a initiator and a bus, and tags all TileLink (TL) transactions. The wgMarker uses the TL `userField` to carry the WID.

The wgMarker has two nodes: a TL Manager node and a TL Control node. The manager node is responsible for marking the TL transaction, while the control node is responsible for configuring the control registers.

For cores that are not wg-aware and other initiators, the only way to access the wgMarker configuration is via a memory-mapped register. It's accessible only by requests marked with the *trusted WID* (see Figure 11).

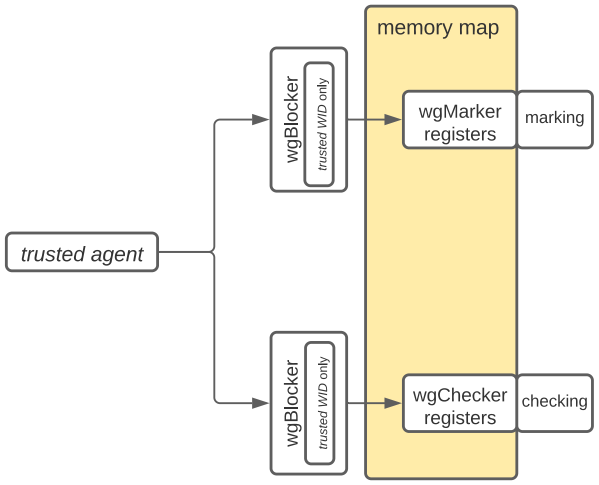

Figure 11: *Trusted Agent wgMarker Access*

A lock bit is available in the wgMarker register; If this lock bit is set, the WID value cannot be changed until the next Core Complex reset.

Figure 12 describes the flow for the marking process, as well as the configuration process for a wgMarker. The wgMarker is memory mapped.

Regarding the configuration process, the wgMarker can be configured through the control node, if the transaction is marked by the *trusted WID* (meaning this transaction must come from the *trusted agent*). This "command" transaction goes through a blocker, rejecting any command not marked with the *trusted WID*. This prevents any command marked with another WID to change the wgMarker configuration.

This configuration affects two main values, the WID value (`WID set`) and the lock bit (`lock set`). If the lock bit is not set (*is lock = 1 ?*, in green), the WID value and the lock bit can be set. Otherwise, if the lock bit is already set, both the WID value and the lock bit cannot be changed, until the next platform reset.

Regarding the marking process, the TL transaction request comes in the block through the manager node and is marked with the WID value (previously set or containing the reset value). The marking operation consists in setting the TL `userField` field with the $\log_2$`NWorlds`-bit WID value. The marked request then leaves the block.

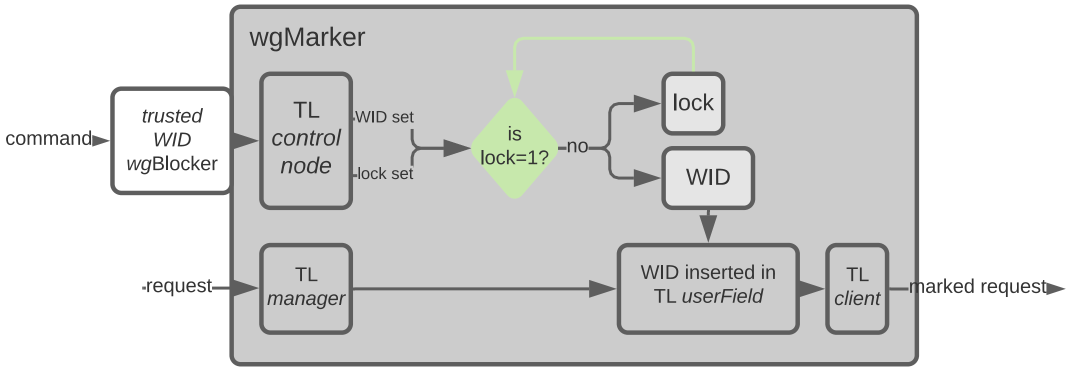

Figure 12: *Generic WGMarker*

There are as many generic wgMarkers as there are initiators (except for the cores, as described in the previous section).

Note

WID change on a non-wg-aware core must be performed in a specific sequence in order to avoid an exception during the WID switch operation. However, it does not practically make sense to change the WID.

6.2.4 WGMarkers for WG-aware Cores

If the wgMarker is integrated within a core (i.e., within the tile), its configuration access is available through CSRs. These CSRs help determine the wgMarker WID value that is used for marking outgoing transactions. Some of these CSRs' accesses depend on privilege mode.

A list of WIDs the core is allowed to use, called `mwidlist`, is provided as wired inputs to the core (similar to the reset vector). The `mwidlist` may be hardwired or connected to a boot-time-programmable register. The wgMarker WID value can be set to any of the WID values enabled in the `mwidlist`.

The setting of the wgMarker value is performed through the setting of the WID registers. Beyond the WID value for the M-mode, which is set by outside wires via the WG Port and is not accessible from the core software, there are up to three distinct, similar binary-encoding 32-bit CSRs depending on available privilege modes (`mlwid`, `vslwid`, `slwid`). These CSRs contain the WID value to be set in the wgMarker when the register's corresponding privilege mode is the current privilege mode.

- The value contained in the `mlwid` CSR is loaded in the wgMarker when the core privilege mode is the next lower privilege mode (HS-mode if implemented, else S-mode if implemented, else U-mode)
- The value contained in the `slwid` CSR is loaded in the wgMarker when the core privilege mode is U-mode or VS-mode (provided HS-mode and S-mode are implemented, respectively)
- The value contained in the `vslwid` CSR is loaded in the wgMarker when the core privilege mode is VU-mode (provided the hypervisor extension is implemented)

The M-mode WID value is set by outside wires, in a way similar to the `mwidlist`. As this WID setting is independent from the `mwidlist` setting, the two values may be, for whatever reason, incompatible; consequently, the core wgMarker WID value is set to zero causing the core to trap on any memory access. It is up to the implementation to guarantee that these values are not incompatible.

The `mwidlist` is not directly visible to M-mode software. However, M-mode software can determine which WIDs can be used by leveraging the fact that the `mlwid` CSR is Write-Any-Read-Legal (WARL); software can attempt to write a WID value to `mlwid`, then read `mlwid` back to see if the write succeeded.

WID Settings

The considered scenario for the WID change is that an upper privilege mode changes a lower privilege mode's `Xwid` register. As such, this specification considers that `X` mode cannot set `Xwid`.

- For M-mode only cores, there is no CSR. The M-mode value is set by outside wires.
- For M/U cores, there is one CSR, `mlwid`, used by the M-mode firmware to set the U-mode WID value
- For M/S/U cores, there are three CSRs:
    - `mlwid`, used by the M-mode firmware to set the S-mode WID value
    - `mwiddeleg`, used by the M-mode firmware to list the WIDs the S-mode firmware can set on its side
    - `slwid`, used by the S-mode firmware to set the U-mode WID value
- For cores with the hypervisor extension (M/HS/VS/VU), there are five CSRs:

  - `mlwid`, used by the M-mode firmware to set the HS-mode WID value
  - `mwiddeleg`, used by the M-mode firmware to list the WIDs the HS-mode firmware can set on its side
  - `swiddeleg`, used by the HS-mode firmware to list the WIDs the VS-mode firmware can set on its side
  - `slwid`, used by the HS-mode firmware to set the VS-mode WID values
  - `vslwid`, used by the VS-mode firmware to set the VU-mode WID value

Delegation

The purpose of delegation is to have a privilege mode `X` delegate to privilege mode `X-1` the right to set the WID value in the `[X-2]wid` CSR for the privilege mode `X-2`.

- For M-mode only cores and M/U cores, there is no notion of delegation and there is no delegation CSR
- For M/S/U cores, the typical use case is to have M-mode (i.e., the security monitor) delegate to S-mode (i.e., the RTOS) the ability to set the WID for U-mode (i.e., the tasks).

  If the S-mode firmware needs to determine which WID values it can use, it can use the fact that `slwid` is a WARL register; attempt to write a value to `slwid`, then read back `slwid` to see if the write succeeded.

  There is one CSR dedicated to delegation, the `XLEN`-bit `mwiddeleg` CSR that controls delegation from M-mode to S-mode.
- For M/HS/VS/VU cores, the typical use case is to have M-mode (i.e., the security monitor) delegate to HS-mode (i.e., the type-1 hypervisor) the ability to set the WID for the different containers, gathering S-mode (i.e., the OS) and the U-mode (i.e., the tasks).

  If the HS-mode firmware needs to determine which WID values it can use, it can use the fact that `slwid` is a WARL register; attempt to write a value to `slwid`, then read back `slwid` to see if the write succeeded.

  There are two CSRs dedicated to delegation, the `XLEN`-bit `mwiddeleg` and `swiddeleg` CSRs. `mwiddeleg` controls delegation from M-mode to HS-mode. `swiddeleg` controls delegation from HS-mode to VS-mode and exists only if the hypervisor extension is implemented.

WID Delegation Use Case

Consider a platform with `NWorlds=8` (so the *trusted WID* is 7) and a M/S/U core. The software architecture has a security monitor in M-mode, an RTOS in S-mode, and tasks in U-mode. The configuration is performed as follows:

1. The `mwidlist` contains the worlds from 1 to 7, meaning these values can be handled in M-, S-, and U-mode (at this stage of the configuration process)
2. The core's M-mode WID value is set to the value 7 (the *trusted WID*) by outside wires
3. The `mlwid` is set to 1 (the WID for the RTOS) in M-mode
4. The `mwiddeleg` is set to 2, 3, 4, 5, and 6 (the possible WID values that S-mode can set for U-mode) in M mode. The `slwid` (the WID value for U-mode) can be set in S-mode with one of those values.

Thus, the RTOS is able to change the `slwid` value among the values 2–6, but cannot change to 7 or 1 as neither of them are set in the `mwiddeleg`.

WGMarker Value Computation

Beyond these CSRs, a core wgMarker must be designed within the tile in order to process the transaction outgoing the block and mark this transaction with the WID value. This WID value is inherited from the CSRs based on the privilege mode. The following figures provide an overview of the CSRs described above and their integration in the core.

- For M-only cores

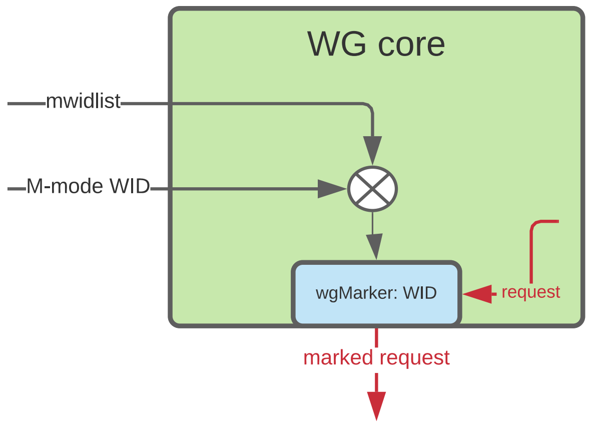

Figure 13: *WorldGuard M-only Core Marker CSRs*

- For M/U cores

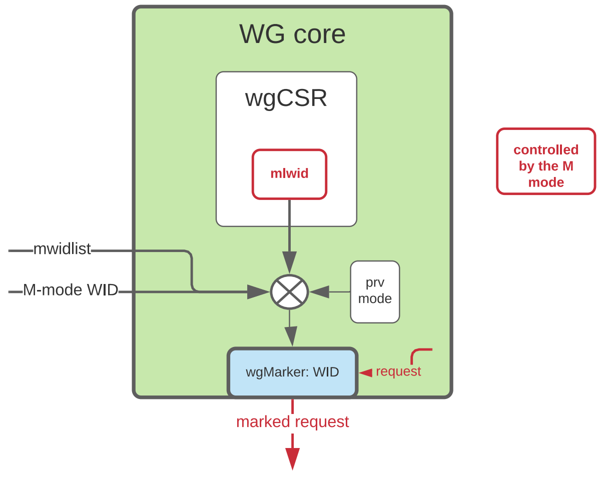

Figure 14: *WorldGuard M/U Core Marker CSRs*

- For M/S/U cores

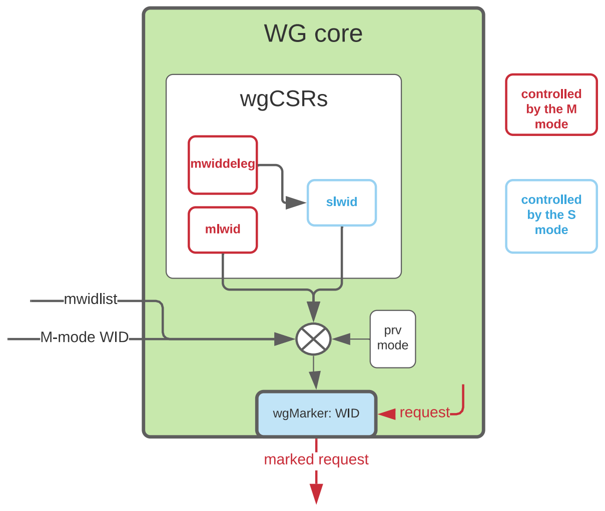

Figure 15: *WorldGuard M/S/U Core Marker CSRs*

- For M/HS/VS/VU cores

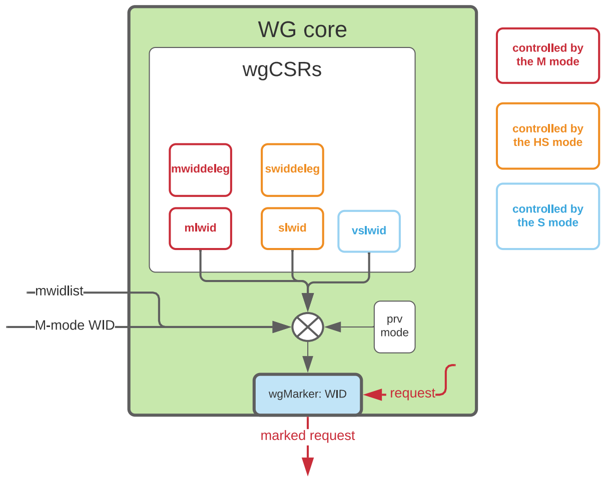

Figure 16: *WorldGuard M/HS/VS/VU Core Marker CSRs*

6.2.5 WGCheckers

A wgChecker is used to protect a resource from illegal accesses. Figure 17 describes its mechanisms and functioning. The wgChecker's main role is to analyze the marked requests sent to resources (TL initiator node) and grant or not grant the request access. This analysis is based on request fields such as the WID value, the access form (Read, Write), and the targeted address in the resource. For this analysis, configuration registers, named slots, define access rules for regions of the protected resource; these rules reflect the request fields, i.e., authorized WID value for specific attributes on some addresses.

If the analysis is satisfying, i.e., a slot matches the request field, the access is granted, and the request is approved and sent to the resource. If the analysis is not satisfying, meaning there is no slot matching the combination of the address, the attribute and the WID, two consequences are possible:

- WGChecker error registers are set with the rejected fields: the WID, the attribute and the address
- A `denied` response is sent back to the request emitter and an interrupt is triggered. Note that these events are possible if and only if the interrupt bit is set. Otherwise, the interrupt is not emitted and the `denied` is not sent back. Not triggering an interrupt is important in the case of a speculative instruction fetch that could have performed an illegal access.

Making this analysis possible requires the wgChecker registers to be set. This is done using a memory-mapped access blocked to the *trusted WID*.

Beyond the simple interrupt bit set, the configuration of a complex wgChecker, a wgPMP, relies on several slots that represent the various rules defined for this resource. These are these rules that are used during the analysis.

The wgPMP contains a number of slots. A slot can be set if and only the slot lock bit is not already set to 1. If this bit is not yet set, the configuration can define start and end addresses, delimiting a region in the memory, a WID list (i.e., the worlds that have access to this region), and the authorized access type for each of the WIDs in the list (R, W).

The wgCheckers accept or reject transactions based on WID. In case of failed transactions, they capture information in local registers.

In case of rejected transactions, the responses are:

- For loads: return zeroed payloads, may respond denied access
- For stores: acknowledged (no access denied response), but not sent downstream
- For AMO: return zeroed payload, but otherwise ignored
- For CMO: acknowledged, but otherwise ignored

The wgCheckers' interrupts can be masked by whatever interrupt controller they are attached to.

The wgCheckers are used to filter the requests sent from a initiator, marked with a WID value, before transferring the request to the sought resource (memory, peripheral).

The wgCheckers may vary in complexity, depending on their features:

- Their simplest form, named wgBlocker, is for controlling wgMarkers or more complex wgCheckers; their control port is blocked on the *trusted WID* in a hardwired manner
- The intermediate form, named wgFilter, hosts a WID list register for granting or not granting access to the resource
- Finally, the most complex and flexible form, named wgPMP, is a block very similar to a core PMP (as defined in [1]), where the different slots defining the memory portions also contain the authorized WID values and access attributes

The wgChecker configuration access is memory mapped. It is usually controlled by a wgBlocker, so accessible only by requests marked with the *trusted WID*. Nevertheless, variations can authorize a more flexible policy for the configuration access, e.g., a specific WID can be granted the access for configuring a wgChecker protecting a resource dedicated to this WID value.

It is important to note that if there is a wgChecker before the output port, there’s no need for a WID conversion at the output (as there is no WID anymore), while the lack of wgChecker before the output port requires a conversion effort **and** maybe a wgChecker. For sake of simplicity, we don’t describe any conversion block in this document.

WGPMPs

The wgPMP block is the assembly of three kinds of registers, the interrupt register, the error registers and the wgPMP slots. The wgPMPs' regions defined in the slots may overlap each other.

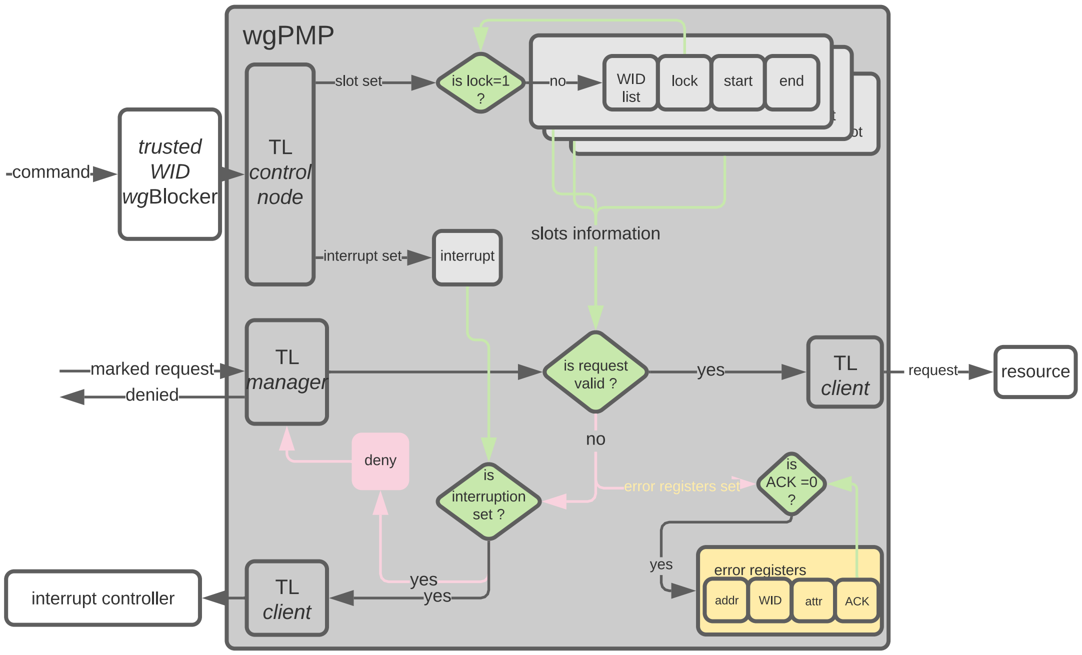

Figure 17: *TileLink WGPMP*

WgPMPs slots are made of an address `start`, `end`, `lock` and authorized WIDs list fields.

Additionally, a simple register for managing the interruption option is added to the slots.

WGFilters

Some resources, such as peripherals, may not require the complexity of the wgChecker. In order to save space, a simpler wgChecker block is proposed (see Figure 18), called wgFilter.

Compared to the complex wgChecker, this simple wgChecker neither has slots nor memory portions. There's only a list of authorized WIDs (`WIDlist`) with their access attributes (R, W), that is compared with the marked request. Error management and interrupt management are similar to the generic wgChecker.

The marked request is analyzed against the WID list. If the analysis is satisfying, i.e., if the WID and access type of the request match the WID list and their attributes, the request is approved and sent to the resource (`TL client`). If the analysis is not satisfying, the error and interrupt management is similar to the generic wgChecker. The configuration is simpler than for a generic wgChecker, as there's only three registers to set, the WID list and their access attributes, the lock bit and the interrupt bit.

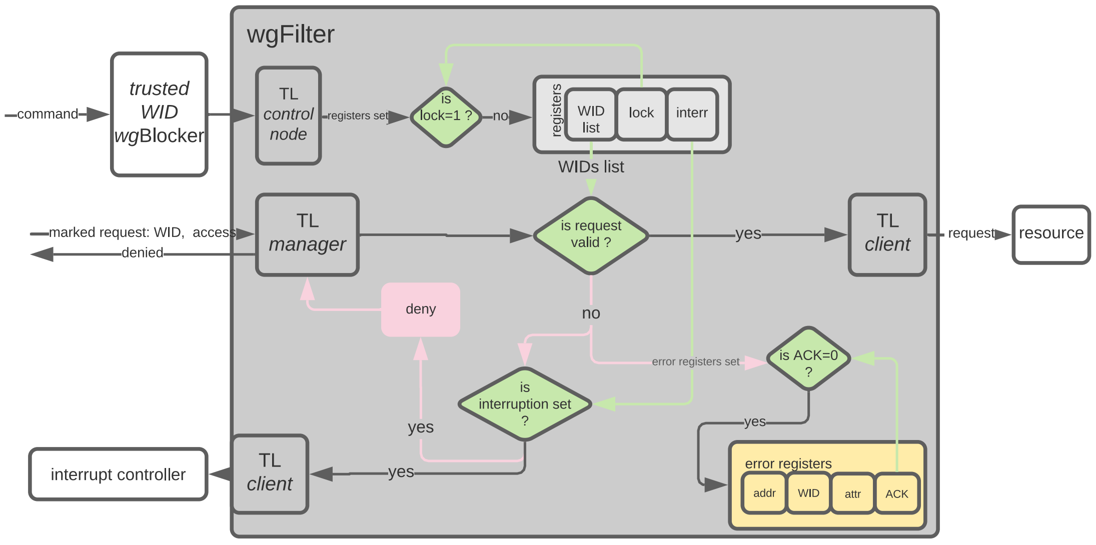

Figure 18: *TileLink WGFilter*

The wgFilters for simpler controls (peripherals, TIMs, L2 I/O configuration) are considered with the specific purpose of simply granting or not granting access to a target resource, without any granularity on subregions. This access control is performed by defining a WID list, similar to the wg-aware cores' WID lists registers. This list contains a set of WID values authorized to access the protected resource.

The wgFilter is the assembly of three registers, the interrupt register, the error registers and the authorized WIDS list.

WGBlockers

The simplest form of the wgCheckers is named a wgBlocker. It is used to control wgMarkers or more complex wgCheckers; their control port is blocked on the *trusted WID* in a hardwired manner, and there is no way to configure them. A wgBlocker checks a marked transaction against a hardcoded value, usually the *trusted WID*. It has to be placed before any wgMarker, wgFilter, or wgPMP.

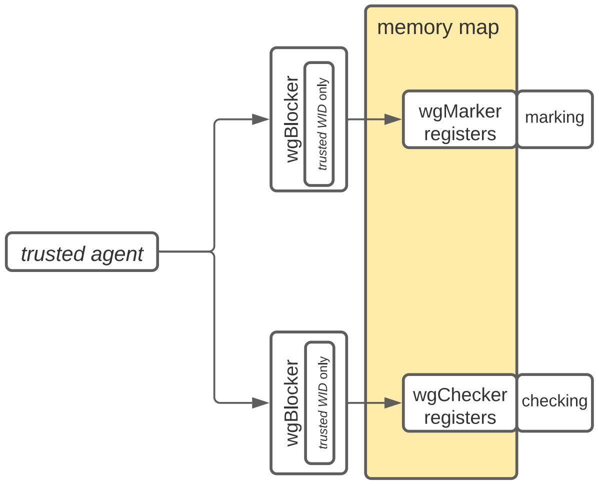

Figure 19: *WGBlocker*

Error Registers

Any access to the protected resource may be rejected by the wgChecker. Depending on rules, hardcoded or not, defined in the wgCheckers. In case a violation is detected, it is possible to send a denied access notification and also to generate an interrupt signal.

So, the proposed solution is made of two items:

- First, some registers, the error registers, containing important information are included in every wgFilter and wgPMP (the wgBlocker does not contain any error register). Looking for error information is performed by parsing each wgChecker. These memory-mapped registers are blocked for the *trusted WID* value only. If the error-initiating core interrupt handler needs the error information, it has to query the *trusted agent* for access and transmit the information to this handler.

  In case a *trusted core* is present, the global interrupt on the Platform-Level Interrupt Controller (PLIC) must also be set to trigger an interrupt to the *trusted core*. This global interrupt has a lower precision than the local interrupt.
- The wgChecker is also connected as an interrupt source to the interrupt controller, which can be the PLIC or the Core-Local Interrupt Controller (CLIC), or any other interrupt controller. As a good practice, it is used to notify the *trusted agent*, if the *trusted agent* is registered to this interrupt (the wgChecker interrupt configuration bit represents that).

6.2.6 Locking Policy

The WorldGuard configurations in the wgMarkers and in the wgCheckers can be locked (e.g., after initial configuration) or kept dynamic for in-session re-configuration by the *trusted agent*.

If a lock is set, there is no way, even for the *trusted agent*, to modify the related setting until the next platform reset.

Locking any WorldGuard block or register of a block on the platform is initially determined by the WorldGuard configuration used at startup. Any block or register not locked at this startup can be later locked by the *trusted agent* at any time.

6.3 Memory Map

The following table describes the blocks and their usage:

Block	**Usage**	**Comment**
wgMarker	Mark a request transaction at the exit of an initiator, using the WID value. To be placed after a non-wg-aware initiator.	Memory-mapped, blocked on the *trusted WID* only
Core wgMarker	Mark a request transaction at the exit of the core, using the WID value. To be placed in a wg-aware core tile.	CSRs are used to manage the wgMarker WID value
wgPMP	Check a marked transaction based on a set of rules (slots) defined with address ranges, WID values and access kind. To be placed before a memory, a DMA.	Error information is restricted to *trusted WID* access
wgFilter	Check a marked transaction versus a list of WID/access types. To be placed before a peripheral, an interrupt controller, a crypto block.	Targets the peripherals and simple access control
wgBlocker	Check a marked transaction against a hard-coded value, usually the *trusted WID*. To be placed before a wgMarker, a wgFilter, a wgPMP	Targets any WorldGuard block on the platform

Figure 20 describes the components of wgCheckers (wgPMPs and wgFilters).

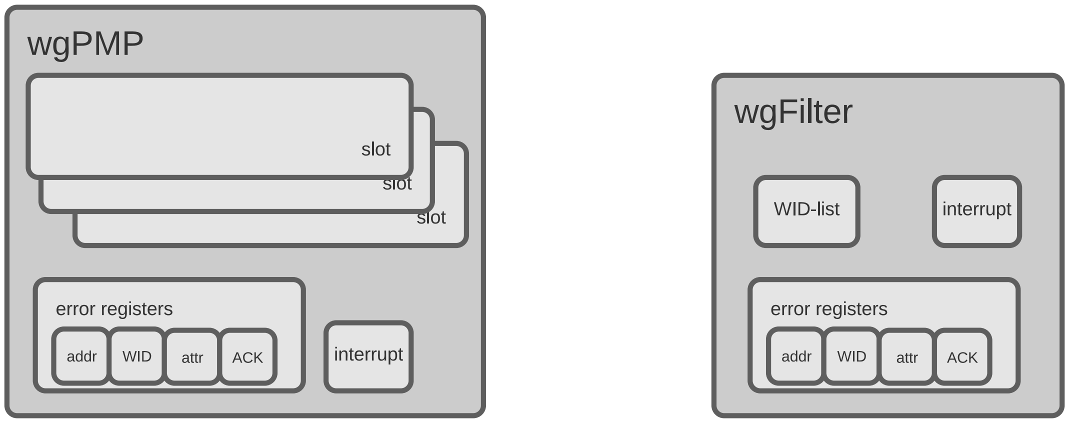

Figure 20: *Inclusion of Sub Blocks in the WGCheckers (WGFilters and WGPMPs)*

6.4 Hardware Interface

There exist three interfaces for the WorldGuard blocks:

- The configuration interface, for setting the block registers
- The input interface, where the request enters into the block
    - On wgMarker, the request enters unmarked
    - On wgChecker, the request enters marked
- The output interface, where the request exits from the block
    - On wgMarker, the request exits marked
    - On wgChecker, the request exits unmarked

6.4.1 Initiator Interface

The wgMarker block initiator interface is used to add the WID value, stored in the wgMarker on the bus.

The wgChecker block initiator interface is used to process the marked request from the bus.

6.4.2 Target Interface/Control

Except the wgBlocker, any WorldGuard block can be configured using its configuration interface; it is the block target interface, accessible only to the *trusted agent* or to authorized WIDs. The memory-mapped registers' access is granted only to the requests marked with the *trusted*

WID. For the wg-aware core wgMarkers, the CSRs' access is granted only to the firmware running on the core the CSRs belong to.

6.5 Software Interface

On wg-aware cores, the software has access to the WorldGuard core wgMarker CSRs it runs on. This access depends on the privilege mode the software runs in, and also the values of the `mwidlist` and `Xwiddeleg` registers.

The way the `mwidlist` and the core M-mode WID value are set by the output wires is various and may be accessible to some software at reset.

Aside from these CSRs, the other WorldGuard blocks configuration registers are memory-mapped, and their access is controlled (only on the *trusted WID* , but there can be exceptions). Therefore, any firmware able to run in the *trusted WID* can access and modify these registers.

6.5.1 Access Rights

The memory-mapped registers' access is (generally) constrained by the *trusted WID* value.

For WorldGuard CSRs (on the core-local wgMarker), access is constrained by privilege modes.

6.5.2 Changing WID

Non-wg-aware Cores

The WID change in a non-wg-aware core is managed by the *trusted agent*.

WG-aware Cores

On wg-aware cores, the WID change that occurs when the privilege mode changes is hardware-managed, based on the new privilege mode N value (core M-mode WID value and `mlwid, vslwid, slwid` registers).

The following operations **are not possible**:

- The M-mode WID value change
- The X-mode WID value change by X-mode firmware

6.6 Software Configuration and Usage

6.6.1 Default Configuration

On reset, the M-mode firmware starts by running in the "reset value world", which is a separate input to the core (input by outside wires via the WG Port, as described in Section 6.2.1).

Note that at least one wg-aware core wgMarker WID value must be set to the *trusted WID* and its `mwidlist` has the *trusted WID* bit set.

6.6.2 Secure Boot Process and Initial WorldGuard Configuration

The secure boot process is used to set the initial WorldGuard configuration in trusted conditions as this configuration is highly critical for the platform assets' management.

A secure boot code in a ROM is required as the Root of Trust (RoT).

At reset, all the core(s) start and jump to the ROM. After completion of any setup and RoT task, execution of the secure boot code begins, more particularly the first mutable code, a.k.a. the First Stage Boot Loader (FSBL). In the dedicated *trusted core* use case, the *trusted core* is in charge of operation. In the multi-core without a *trusted core* and single-core use cases, it can be any of the cores that are assigned the *trusted WID* to its M-mode WID value. This core is then considered as the *trusted core*, having the *trusted agent* privileges.

The *trusted core* completes the secure boot code execution in ROM by authenticating the FSBL. The FSBL (among potential other tasks like secure update, PCIe setup, memories' setup) determines where the WorldGuard configuration is located, e.g., a hardcoded address or an OTP-stored address. The potential complexities and flexibility requirements justify that it is up to the FSBL to manage the WorldGuard configuration and not the ROM code.

Considering this address and other constraints (need for RAM memory for its execution), the FSBL initiates a minimal configuration.

As the WorldGuard blocks configuration "file" is now available, the FSBL interprets it and sets all the WorldGuard blocks. Once this configuration is set, potentially including locks, the FSBL wakes up the other cores and application software execution can begin.

7

Error Management

This section describes error management, in both data and instructions accesses and considers the specific case of speculative accesses.

7.1 Incorrect CSR Usage

Some errors cases considered when using a CSR:

- Trying to access a non-existing CSR or violating the CSR access rules raises a trap (regular CSR policy)
- Providing an incorrect WID, by e.g., setting the `mlwid` CSR with a value not contained in the `mwidlist`, results in the `mlwid` value being set to zero

7.2 Illegal WID Management

There are two kinds of accesses, instruction and data accesses. Any wgChecker contains an interrupt selection bit that determines if a denied access is notified and/or an interrupt is raised to another interrupt controller.

There are two policies regarding interrupts:

- For instructions errors, the exception handler running on the faulty/initiating core, may request error information from the *trusted agent*. Only the *trusted agent* is able to access the wgCheckers error registers
- If the core exception handler cannot manage the WG errors, the interrupt controller can be selected and then the interrupt is raised and the interrupt handler on the faulty/initiating core manages the error handling in the same way

7.3 Speculative Accesses

When speculative accesses are performed, it may happen that speculatively accessed instructions' addresses do not belong to the core world and so, an illegal access is performed. This illegal access must be ignored, as it is not an intentional access but only a consequence of the microarchitecture.

For instruction access requests, the wgChecker returns the instruction as zero to the core. As this zero instruction is ultimately not executed, there is no exception. For data access requests, the wgChecker returns the data as zero. This zeroed value may not be detected by software as is, a zero value may be a licit value. For this reason, it is recommended to use early control mechanisms such as the core PMP and/or the MMU for controlling these kinds of errors.

Thus, speculative accesses are safe in regard to WorldGuard.

8

Impact on Generic Hardware Blocks

Several generic hardware blocks on the platform are impacted by the WorldGuard solution. This is usually because these hardware blocks have not been specified with isolation requirements. Therefore, their generic implementation and use "as is" cannot comply with the WorldGuard security rationale due to access control violations.

There are two types of consequences, depending on if these blocks are either implicitly or explicitly guarded. If they’re implicitly guarded, there’s no requirement for any modification on the block, but some specific software process is required at WID value configuration change. If they’re explicitly guarded, these hardware blocks must either be modified or equipped with a wgChecker on their configuration interface, if any, or both.

Details on applicable hardware blocks are given in the microarchitecture specification, but a simple description is also given here as an overview. These sections also consider some representative hardware blocks that are not impacted by the WorldGuard solution.

8.1 Cores

Non-wg-aware cores are not modified. For wg-aware cores, CSRs have been added.

8.2 L1 Cache

On non-wg-aware cores, the L1 cache (when applicable) is not modified, but each time the core wgMarker WID value is changed, the L1 cache must be flushed (however, this WID change is unlikely to happen).

On wg-aware cores, the L1 cache must be modified to include the WID in the cache line tag data. Furthermore, the eviction policy must also consider this WID value as a reason for eviction if the proposed WID value is different from the stored WID value.

Note

It is a very similar situation to the L2/LLC Cache block.

8.3 L2 Cache and LLC

When applicable, the L2 cache and further LLC have to be modified for a multi-initiator configuration. The cache lines must be modified by the addition of the WID value in the tag. Therefore, each time the cache line is accessed from a marked transaction, there is a WID field check between the stored WID and the request WID. If they don't match, there is a cache miss.

An illegal access leads to an access denied, and the response that comes back has a zeroed payload. This response is stored in the cache with the same WID value and a zero line. If the line is dirtied in the cache, the eviction write back will be ignored.

Furthermore, as L2 cache and LLC may have a (target) configuration interface (e.g., way locking, LIM configuration), this configuration interface must be protected by a wgChecker, in order to filter the configuration modifications requests based on the authorized WIDs.

8.4 TIM

When applicable, a Tightly-Integrated Memory (TIM) can be instantiated close to a core, as an alternative to L1 caches. But even though this TIM grants fast local access with the core, the TIM can be also accessed by other cores via the bus. A control must be set for these two kinds of accesses. On bus access, it has the form of a wgChecker able to filter the authorized WIDs. On local access, as the TIM is not explicitly guarded, the core PMP can play the access rights controller role.

Depending on the software architecture and the TIM size, the wgChecker may be a wgFilter or a wgPMP.

8.5 Bus

There is no required modification on TileLink buses because of WorldGuard. On TileLink, the WID is transmitted using the already existing `userField` field. The WID is propagated with the request through the TileLink to the concerned target.

On other bus protocols, such as AXI, a field similar to `userField` can be used for WID transport.

8.6 CLIC/PLIC

The Core-Local Interruptor (CLINT) and CLIC contain a register named `mtimecmp` that is used for comparing (and potentially triggering) with `mtime` for a time-based interrupt. Typically, the software sets the value of `mtimecmp` with the value `mtime` *ticks* where the *ticks* value is the duration of the task to be interrupted. When `mtime` reaches this value, the interrupt will trigger.

The security issue is that the `mtimecmp` register for one hart can be configured by another hart. The WorldGuard solution proposes an interrupt management scheme where interrupts handlers can be linked to a world and its WID. That is, WorldGuard protects access to the CLINT and the CLIC by adding a simple configuration wgFilter that filters the `mtimecmp` registers based on the WID values corresponding to the authorized harts. These WID values are set up by the *trusted core*.

8.7 Debug Module

Debug mechanisms must be able to conform to the WorldGuard configuration and if required, debug must be prevented for certain worlds, for their software and their resources (e.g., memories).

This scenario is typical of a software architecture where some portions of the code, for example the TEE or the secure enclave, are not accessible to the application developer and so must not be available to the debug module, while the application that is currently developed must be available to the debug module.

8.7.1 WG-aware Cores

On wg-aware cores, some mechanisms have been implemented to address isolation requirements.

8.7.2 Debug Security Level

Beyond the world control, there is no limitation to the debug features. It is therefore strongly recommended to use the secure debug mechanism (contact SiFive for more information) to control system-level debug activation.

Nevertheless, more aggressive options could consist in having a `DBG_DISABLE` input to disable debug functionality when set. This input can be connected to a fuse or a debug authentication signal.

8.8 SBA

The System Bus Access (SBA) implements a bus initiator that connects with the bus crossbar to allow access to the device’s physical address space without involving a core to perform accesses.

The SBA behaves like a DMA initiator in that it can read or write any memory accessible in the Core Complex. Without any control, it then can have full access to the memory. Therefore, the recommended approach to use it with a wgMarker. This wgMarker WID value is a specific value, not necessarily common to any of the worlds, used by the cores. It is the wgCheckers' configuration that determines what the SBA can access, independently of what the software running on the cores can access (typically, an SBA can be used to monitor some portions of the memory not belonging to the software).

9

10 Integration Examples

9.1 In/Out of the Core Complex

9.1.1 Front Port

An example is to convert an AXI transaction into a WorldGuard-marked transaction. A configurable mapper block is required to convert the AXI AxUSER or AxPROT fields into one (or several) WIDs.

9.1.2 Exit Ports

Similar to the Front Port, but the other way around.

9.2 Within the Core Complex

9.2.1 DMA

A DMA can be a simple DMA and, in that case, is considered as a non-wg-aware initiator.

A more flexible DMA can be a DMA with multiple channels, and any of them can have a distinct WID. The management of jobs is WID-dependent, so a wgFilter is set in front of each channel jobs' queue. If the queue is unified, any of the jobs holds a WID field. Finally, the overall DMA configuration, including the channel WID allocation is protected by a wgFilter or a wgBlocker.

9.2.2 HCA

A WorldGuard-aware Hardware Cryptographic Accelerator (HCA) within the Core Complex is able to perform independent computations based on the WID value. This allows the HCA to define WID-based contexts hosting (secret) keys and data.

10

References

[1] RISC-V Privilege Specification, Available: https://riscv.org/technical/specifications/

[2] RISC-V Debug Specification, Available: https://riscv.org/technical/specifications/

[3] Keystone Enclave Architecture, Available: https://keystone-enclave.org/