Bachelorarbeit

Analyse der Echtzeitfähigkeit von Micro-ROS und FreeRTOS am Beispiel einer Robotersteuerungssoftware

An der Fachhochschule Dortmund
im Fachbereich Informatik
Studiengang Technische Informatik
erstellte Thesis
zur Erlangung des akademischen Grades
Bachelor of Science
B. Sc.

Xu, Zijian
geboren am 25.09.1998
7204211

Betreuung durch:	Prof. Dr. Christof Röhrig
	M. Sc. Alexander Miller
Version vom:	Dortmund, 16. März 2025

Kurzfassung

Diese Arbeit analysiert die Echtzeitfähigkeit von Micro-ROS und FreeRTOS am Beispiel einer Robotersteuerungssoftware. Ziel ist es, die Performance beider Systeme im Hinblick auf Ausführungszeiten, Ressourcenverbrauch und Echtzeitverhalten zu vergleichen.

Die Analyse beginnt zuerst mit der vollständigen Umstellung der bestehenden Robotersteuerungssoftware von Micro-ROS auf FreeRTOS. Anschließend wird die Data Watchpoint and Trace Unit (DWT) Zur Analyse eingesetzt, um eine zyklengetreue Erfassung des Programmlaufs zu ermöglichen.

Abschließend wird das Ergebnis evaluiert, welches unter anderem die Ausführungszeiten von FreeRTOS-Prozessen, zeitkritischen Funktionen sowie das Verhältnis von Ausführung zu Leerlaufzeit umfasst. Die Ergebnisse sollen Einsichten darüber geben, inwieweit Micro-ROS und FreeRTOS für Echtzeitanwendungen in der Robotik geeignet sind und welche Vor- oder Nachteile die jeweiligen Systeme bieten.

Abstract

Inhaltsverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

DWT Data Watchpoint and Trace Unit

RTOS Real-Time Operating System

ROS 2 Robot Operating System 2

DDS Data Distribution Service

Einleitung

Die vorliegende Arbeit beschäftigt sich mit der Analyse der Echtzeitfähigkeit von Micro-ROS und FreeRTOS am Beispiel einer Robotersteuerungssoftware. Ziel ist es, die Performance beider Systeme hinsichtlich der Ausführungszeiten, Ressourcenverbrauch sowie Echtzeitverhalten zu untersuchen, um ihre Eignung für Roboteranwendungen zu bewerten.

Die Arbeit beinhaltet schwerpunktmäßig die Entwicklung einer Methode zum Profiling der Steuerungssoftware eines mobilen Roboters. Dabei wird zunächst die bestehende Firmware, die auf Micro-ROS basiert, im Rahmen dieser Arbeit auf FreeRTOS portiert. Anschließend wird die Methodik zur Generierung von Profiling-Daten für die Analyse festgelegt und implementiert, und die resultierende Ergebnisse evaluiert.

Zu Beginn wird ein Überblick über die grundlegenden Konzepte gegeben. Darauffolgend werden die Implementierungen detailliert beschrieben. Abschließend werden die erzielten Ergebnisse vorgestellt und bewertet, und es wird ein Ausblick auf weitere Anwendungsmöglichkeiten und Optimierungspotenziale gegeben.

1 Hintergrund

Die vorliegende Bachelorarbeit hat zum Ziel, die Robotersteuerungssoftware, die derzeit auf Micro-ROS basiert, auf FreeRTOS zu portieren, um einen vergleichenden Leistungsanalyse zwischen beiden Plattformen durchzuführen. Beide Systeme sind für die Steuerung eines mobilen Roboters auf einem Cortex-M7 Mikrocontroller von Arm konzipiert, unterscheiden sich jedoch in ihrer grundlegenden Architektur, was sich auch in ihrer Echtzeitfähigkeit und Ressourcennutzung widerspiegelt. Während Micro-ROS auf der Robot Operating System 2 (ROS 2) aufbaut und eine höhere Abstraktionsebene sowie standardisierte Kommunikationsschnittstellen mittels der Data Distribution Service (DDS)-Middleware bietet, basiert Micro-ROS selbst auf FreeRTOS. Die Portierung der Robotersteuerungssoftware von Micro-ROS auf FreeRTOS kann daher als eine Reduzierung der Abhängigkeitsebene betrachtet werden. Dies ermöglicht eine direktere und effizientere Nutzung der zugrunde liegenden Echtzeit-, sowie Speicherressourcen.

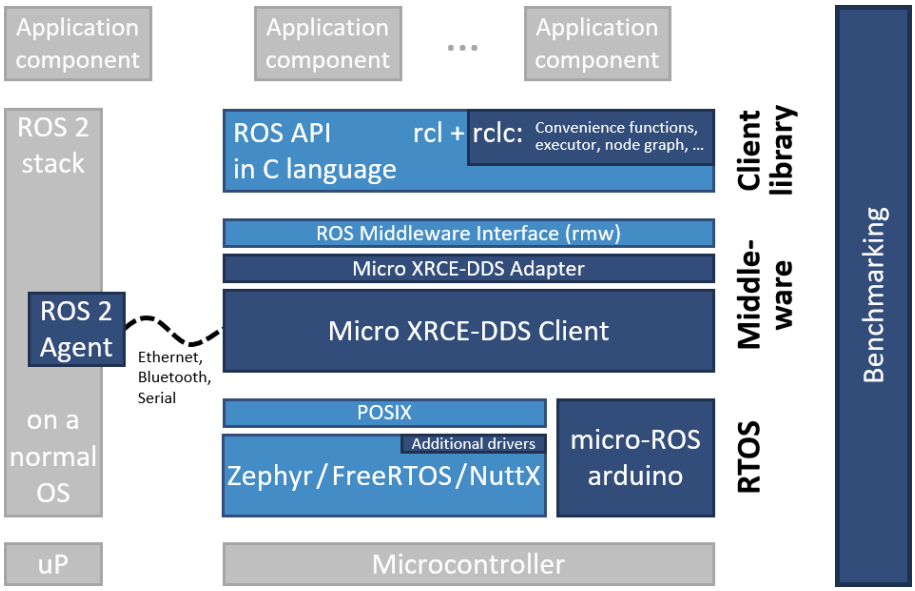

Abbildung 1: Micro-ROS Architektur[Kou23, S. 6]

Nach dem Wechsel zu FreeRTOS wird die Echtzeitleistung der Steuerungssoftware analysiert mit einem besonderen Fokus auf den Overhead, der durch die Micro-ROS-Schicht verursacht wird. Der Vergleich soll aufzeigen, inwiefern FreeRTOS durch die Eliminierung dieser zusätzlichen Abhängigkeit eine effizientere und leichtgewichtige Lösung für kritische Roboteranwendungen darstellt. Dabei soll der Einsatz einer zyklengenaue Messung des Programmablaufs ermöglichen, fundierte Aussagen über die Echtzeitfähigkeit beider Plattformen zu treffen, und den Leistungsgewinn anhand von diesem Beispiel für eine Steuerungssoftware quantitativ zu belegen.

1.1 FreeRTOS

FreeRTOS ist ein Open-Source, leichtgewichtiges Real-Time Operating System (RTOS), das speziell für eingebettete Systeme entwickelt wurde. Es zeichnet sich unter anderem durch deterministisches Verhalten mit Echtzeitgarantie sowie Konfigurierbarkeit der Heap-Allokation aus. Diese Eigenschaften machen es zu einer geeigneten Wahl für Robotersteuerungssoftware, insbesondere wenn Echtzeitanforderungen und effiziente Ressourcennutzung im Vordergrund stehen.

1.1.1 Konzepte

FreeRTOS unterscheidet sich von der Bare-Metal-Programmierung dadurch, dass es eine nützliche Abstraktionsebene für den Nutzer bereitstellt. Diese Abstraktionen ermöglichen es, komplexere Echtzeitanforderungen zu bewältigen, ohne dass der Nutzer diese Funktionalitäten selbst implementieren muss. Beispiele hierfür sind Timer mit konfigurierbarer Genauigkeit (basierend auf den sogenannten Tick [Fred, Frel]), threadsichere Queues sowie Semaphore und Mutexe [Frec]. Diese Komponenten bieten fertige Lösungen für häufige Herausforderungen in der Entwicklung eingebetteter Systeme, sodass der Nutzer solche Werkzeuge nicht mehr selbst anfertigen muss.

Im Fokus dieser Arbeit stehen Queues und „Direct Task Notifications“, die in der Robotersteuerungssoftware zum Einsatz kommen, sowie Semaphore und die sogenannten „Trace Hooks” für die darauffolgende Echtzeitanalyse. Diese Komponenten werden im Folgenden detailliert erläutert.

Queues Queues sind eine der Kernkomponenten von FreeRTOS und dienen der Interprozesskommunikation zwischen Tasks. Sie ermöglichen den threadsicheren Austausch von Daten, und können sowohl zur Datenübertragung als auch zur Synchronisation von Tasks verwendet werden, da dedizierte (Ressourcen-) Synchronisationsmechanismen wie Semaphore und Mutexe sind auf Queues aufgebaut [Fref].

Semaphore Wie bereits kurz erwähnt, sind Semaphore und Mutexe Tools, die den Zugriff auf gemeinsame Ressourcen koordinieren, wobei Semaphore auch zur Synchronisation von Tasks genutzt werden können. Semaphoren sind einfache Mechanismen ohne Unterstützung von Prioritätsvererbung, bei der ein Task mit Besitz von einem *Mutex* mit einer niedrigeren Priorität künstlich auf die gleiche Priorität

des auf den Mutex wartenden Task angehoben wird [Wika]. Wenn eine Ressource dann nur mit einem Semaphor geschützt ist, kann dies zu Prioritätsinversion führen, bei der ein niedriger priorisierter Task die Ressource weiter blockiert, die ein höher priorisierter Task benötigt [Wikb].

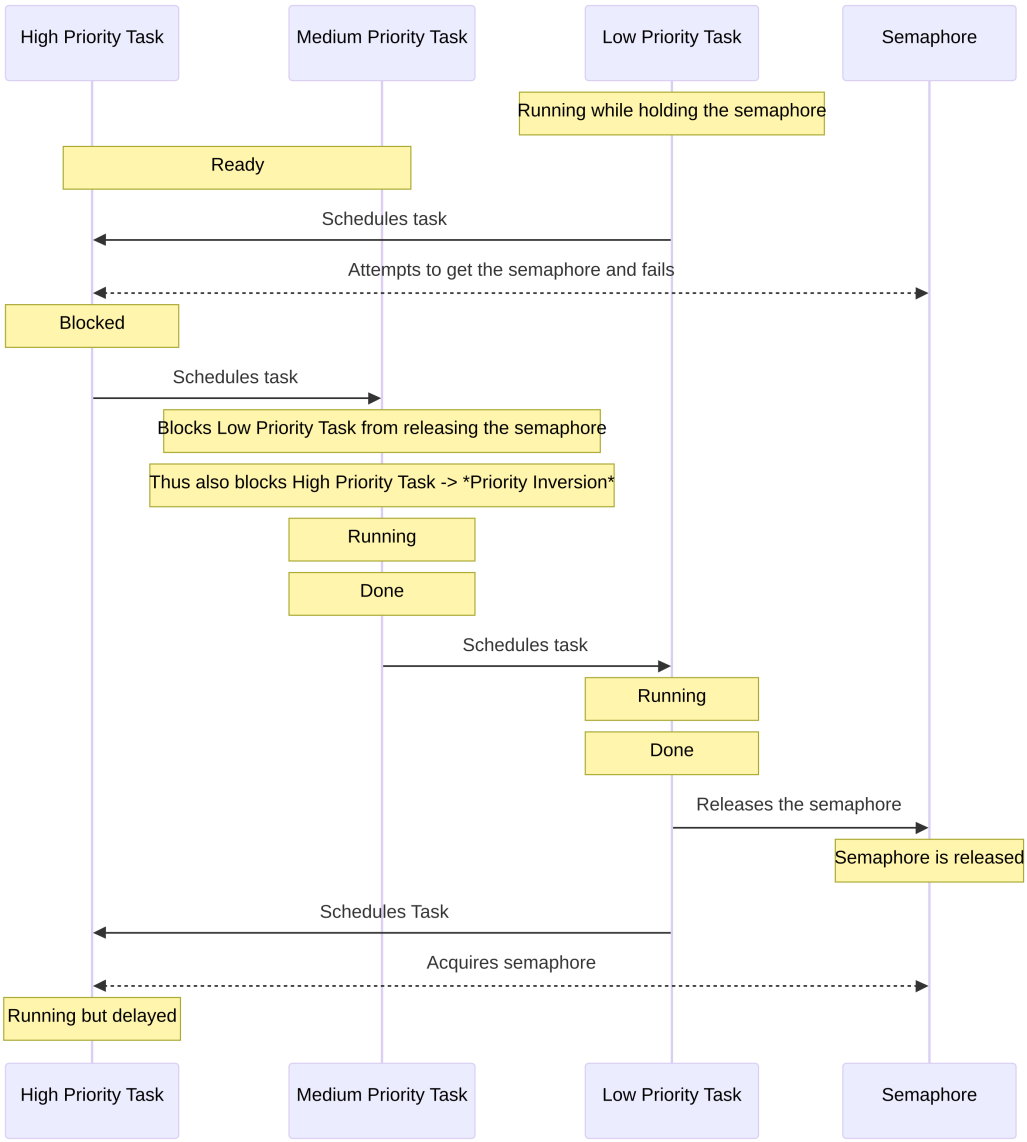

Abbildung 2: Priority Inversion

Mutexe Im Gegensatz dazu sind Mutexe („Mutual Exclusion“) Synchronisationsmechanismen, die Prioritätsvererbung implementieren [Freb]. Wenn ein Task auf einen Mutex wartet, der von einem niedriger priorisierten Task gehalten wird, wird dieser Task temporär auf die Priorität des wartenden Tasks erhöht [Frea], so dass er den Mutex und damit die von dem watenden Task benötigte Ressource so schnell wie möglich wieder freigeben kann.

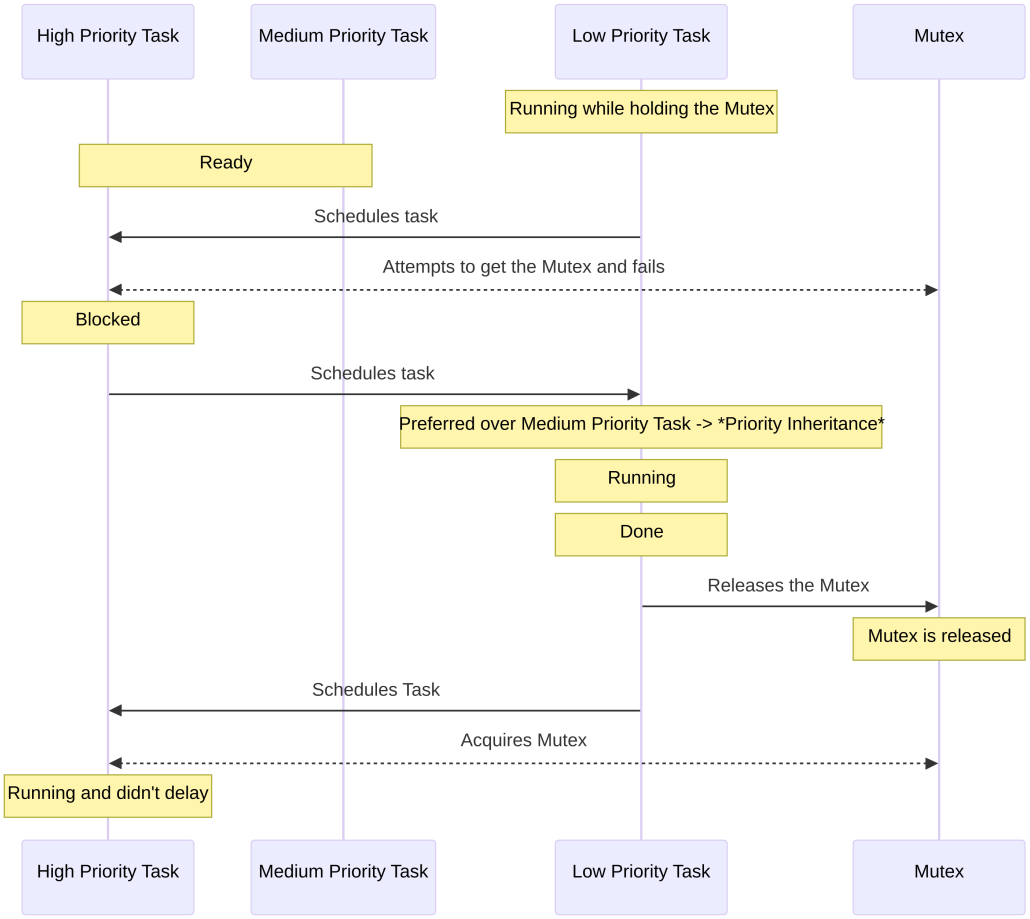

Abbildung 3: Priority Inheritance

Direct Task Notifications Direct Task Notifications sind ein effizienterer und ressourcenschonenderer Mechanismus zur Task-Synchronisation [Freg]. Im Gegensatz zu Semaphoren senden sie direkte Signale an einen Task, ohne die zugrunde liegenden Queues zu benötigen, indem sie einfach einen internen Zähler eines Tasks verändern [Frei]. Analog zu Semaphoren wird mittels Funktionen wie zum Beispiel `ulTaskNotifyGive()` dieser Zähler inkrementiert [Frej], während Funktionen wie `ulTaskNotifyTake()` ihn wieder dekrementieren [Frek]. Das Entblocken eines Tasks mittels Direct Task Notifications soll bis zu 45% schneller sein und benötigt weniger RAM [Freh].

Trace Hooks „Trace Hooks“ sind spezielle Macros von der FreeRTOS-API, deren Nutzung es beispielsweise ermöglicht, Ereignisse im System zu verfolgen und zu protokollieren. Diese Macros werden innerhalb von Interrupts beim Schedulen aufgerufen und müssen immer vor der Einbindung von `FreeRTOS.h` definiert werden [Free].

1.2 Echtzeitanalyse

Um die Echtzeitanalyse der Steuerungssoftware durchzuführen, ist eine Methode erforderlich, mit der beliebige Ausführungsabschnitte der Software flexibel, präzise und threadsicher gemessen werden können. Da die Software multithreaded ist, muss ebenfalls sichergestellt werden, dass die Messungen trotz preemptivem Scheduling sowie Interrupts korrekt und zyklengetreu durchgeführt werden können.

Basierend auf den oben genannten Herausforderungen bietet die Data Watchpoint and Trace Unit (DWT) als eine geeignete Lösung [ARMf]. Die DWT ist ein Debug-Einheit in Prozessoren inklusive ARMv7-M [ARMd], die das Profiling mittels Zähler unterstützen [ARMa]. Ein für diese Arbeit zentraler Teil der DWT ist der Zyklenzähler `DWT_CYCCNT`, der bei jedem Takt inkrementiert wird, solange sich der Prozessor nicht im Debug-Zustand befindet [ARMb]. Dadurch ermöglicht die DWT beispielsweise die Erfassung von Echtzeitaspekten mit zyklengenauer Präzision under normaler Operation [ARMc].

1.2.1 Beispiel: SEGGER SystemView

Ein Beispiel hierfür ist SEGGER SystemView, ein Echtzeit-Analysewerkzeug, das den DWT-Zyklenzähler einsetzt, um Live-Code-Profiling auf eingebetteten Systemen durchzuführen [SEG].

Das SEGGER SystemView nutzt den DWT, indem die Funktion `SEGGER_SYSVIEW_GET_TIMESTAMP()` einfach die hardkodierte Registeradresse des Zyklenzählers zurückgibt, anstatt die interne Funktion `SEGGER_SYSVIEW_X_GetTimestamp()` aufzurufen [SiFa, Arme]. Dies erfolgt, sofern das Macro `SEGGER_SYSVIEW_CORE_CM3` definiert ist [SiFb]. Aus dieser Definition wird gezeigt, dass die Nutzung des DWT für Cortex-M3, M4 sowie M7 Prozessoren aktiviert wird. Daraus könnte sich eventuell auch schlussfolgern lassen, dass der DWT nur auf diesen Prozessoren verfügbar ist.

Aufgrund dieser Eigenschaften der DWT wird die Implementierung für die Echtzeitanalyse auf dieser Komponente aufbauen.

2 Vorbereitung

2.1 Umstellung auf FreeRTOS

2.2 Aktivierung von Caches

3 Abschluss

3.1 Fazit

3.2 Ausblick

Literaturverzeichnis

[ARMa] ARM Holdings: *ARMv7-M Architecture Reference Manual*. https://developer.arm.com/documentation/ddi0403/d/Debug-Architecture/ARMv7-M-Debug/The-Data-Watchpoint-and-Trace-unit/Profiling-counter-support?lang=en. – Zugriff: 14. März 2025

[ARMb] ARM Holdings: *ARMv7-M Architecture Reference Manual*. https://developer.arm.com/documentation/ddi0403/d/Debug-Architecture/ARMv7-M-Debug/The-Data-Watchpoint-and-Trace-unit/CYCCNT-cycle-counter-and-related-timers?lang=en. – Zugriff: 14. März 2025

[ARMc] ARM Holdings: *ARMv7-M Architecture Reference Manual*. https://developer.arm.com/documentation/ddi0403/d/Debug-Architecture/ARMv7-M-Debug/The-Data-Watchpoint-and-Trace-unit?lang=en. – Zugriff: 14. März 2025

[ARMd] ARM Holdings: *Data Watchpoint and Trace Unit (DWT)*, https://developer.arm.com/documentation/ddi0439/b/Data-Watchpoint-and-Trace-Unit/About-the-DWT. – Zugriff: 14. März 2025

[Arme] Arm Holdings: *Data Watchpoint and Trace Unit (DWT) Programmer's Model*. https://developer.arm.com/documentation/ddi0439/b/Data-Watchpoint-and-Trace-Unit/DWT-Programmers-Model. – Zugriff: 14. März 2025

[ARMf] ARM Holdings: *Summary: How many instructions have been executed on a Cortex-M processor?* https://developer.arm.com/documentation/ka001499/latest/. – Zugriff: 14. März 2025

[Frea] FreeRTOS: *Mutex or Semaphore*. https://forums.freertos.org/t/mutex-or-semaphore/14644/3. – Zugriff: 15. März 2025

[Freb] FreeRTOS: *Mutexes*. https://www.freertos.org/Documentation/02-Kernel/02-Kernel-features/02-Queues-mutexes-and-semaphores/04-Mutexes. – Zugriff: 15. März 2025

[Frec] FreeRTOS: *Queues*. https://www.freertos.org/Documentation/02-Kernel/02-Kernel-features/02-Queues-mutexes-and-semaphores/01-Queues. – Zugriff: 15. März 2025

[Fred] FreeRTOS: *The RTOS Tick*. https://www.freertos.org/Documentation/02-Kernel/05-RTOS-implementation-tutorial/02-Building-blocks/03-The-RTOS-tick. – Zugriff: 15. März 2025

[Free] FreeRTOS: *RTOS Trace Feature*. https://freertos.org/

Documentation/02-Kernel/02-Kernel-features/09-RTOS-trace-feature#defining. – Zugriff: 15. März 2025

[Fref] FreeRTOS: *semphr.h*. https://github.com/kylemanna/freertos/blob/125e48f028767ed04a7b27f8ceeec3210a7f1c98/FreeRTOS/Source/include/semphr.h#L138. – Zugriff: 15. März 2025

[Freg] FreeRTOS: *Task Notifications*. https://www.freertos.org/Documentation/02-Kernel/02-Kernel-features/03-Direct-to-task-notifications/01-Task-notifications#description. – Zugriff: 15. März 2025

[Freh] FreeRTOS: *Task Notifications - Performance Benefits and Usage Restrictions*. https://www.freertos.org/Documentation/02-Kernel/02-Kernel-features/03-Direct-to-task-notifications/01-Task-notifications#performance-benefits-and-usage-restrictions. – Zugriff: 15. März 2025

[Frei] FreeRTOS: *tasks.c*. https://github.com/jameswalmsley/FreeRTOS/blob/a7152a969b2b49fce50d759b3972f17bf3b18ed7/FreeRTOS/Source/tasks.c#L213. – Zugriff: 15. März 2025

[Frej] FreeRTOS: *tasks.c*. https://github.com/jameswalmsley/FreeRTOS/blob/a7152a969b2b49fce50d759b3972f17bf3b18ed7/FreeRTOS/Source/tasks.c#L4296. – Zugriff: 15. März 2025

[Frek] FreeRTOS: *tasks.c*. https://github.com/jameswalmsley/FreeRTOS/blob/a7152a969b2b49fce50d759b3972f17bf3b18ed7/FreeRTOS/Source/tasks.c#L3926. – Zugriff: 15. März 2025

[Frel] FreeRTOS: *Tick Resolution*. https://mobile.freertos.org/Documentation/02-Kernel/05-RTOS-implementation-tutorial/02-Building-blocks/11-Tick-Resolution. – Zugriff: 15. März 2025

[Kou23] Koubaa, Anis: *Robot Operating System (ROS) The Complete Reference*. Volume 7. Springer Verlag, 2023. – ISBN 978–3–031–09061–5

[SEG] SEGGER Microcontroller: *What is SystemView?* https://www.segger.com/products/development-tools/systemview/technology/what-is-systemview#how-does-it-work. – Zugriff: 14. März 2025

[SiFa] SiFive: *Segger_SystemView-metal Repository: SEGGER_SYSVIEW_Conf.h*. https://github.com/sifive/Segger_SystemView-metal/blob/cd89dbc2f3d5222d9856780517261ad7a2804805/SystemView/Config/SEGGER_SYSVIEW_Conf.h#L138. – Zugriff: 14. März 2025

[SiFb] SiFive: *Segger_SystemView-metal Reposito-*

ry: SEGGER_SYSVIEW_Conf.h. https://github.com/sifive/Segger_SystemView-metal/blob/cd89dbc2f3d5222d9856780517261ad7a2804805/SystemView/Config/SEGGER_SYSVIEW_Conf.h#L79. – Zugriff: 14. März 2025

[Wika] Wikipedia: *Priority Inheritance.* https://en.wikipedia.org/wiki/Priority_inheritance. – Zugriff: 15. März 2025

[Wikb] Wikipedia: *Priority Inversion.* https://en.wikipedia.org/wiki/Priority_inversion. – Zugriff: 15. März 2025