

Real Time Operating System / RTOS

1 Einleitung

Sie werden in diesem Praktikum die Features eines Real Time Betriebssystems kennenlernen. Ein Betriebssystem kann Ihnen viele Aufgaben abnehmen, wie z.B. das Scheduling von verschiedenen Threads, die quasi-parallel ausgeführt werden können, oder Zugriffe auf gemeinsame Ressourcen mittels Mutexes zu synchronisieren.

- Informationen zum CMSIS-RTOS finden Sie unter <https://www.keil.com/cmsis/rtos>.
- Verwenden Sie für die Ansteuerung der GPIOs die Funktionen aus dem HAL (Hardware Abstraction Layer).

2 Lernziele

- Sie kennen die verschiedenen Scheduling Techniken und können sie in eigenen Programmen anwenden.
- Sie können mit Hilfe von Signals und Mutexes den Programmablauf synchronisieren.

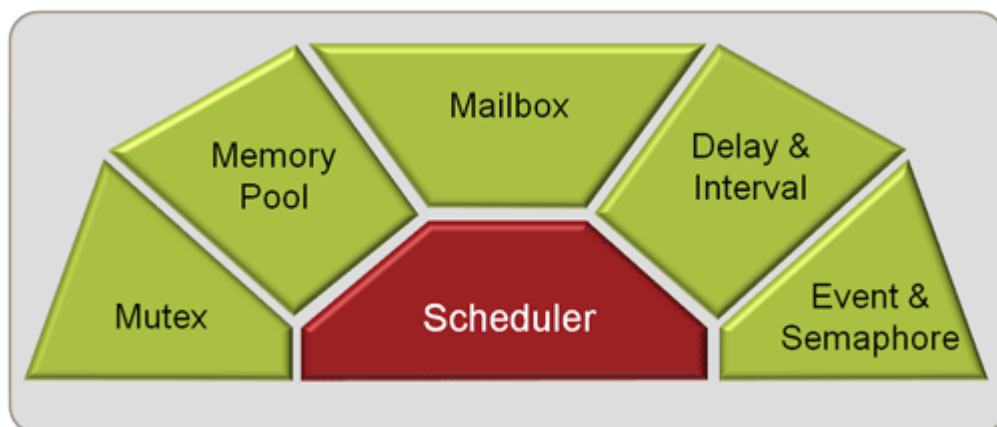


Abbildung 1: CMSIS RTOS Features

3 Multiprojekt

In Keil kann ein sogenanntes Multiprojekt erstellt werden. Ein Multiprojekt ist eine Zusammenfassung mehrerer Keil Projekte, im Idealfall logisch zusammengehörend.

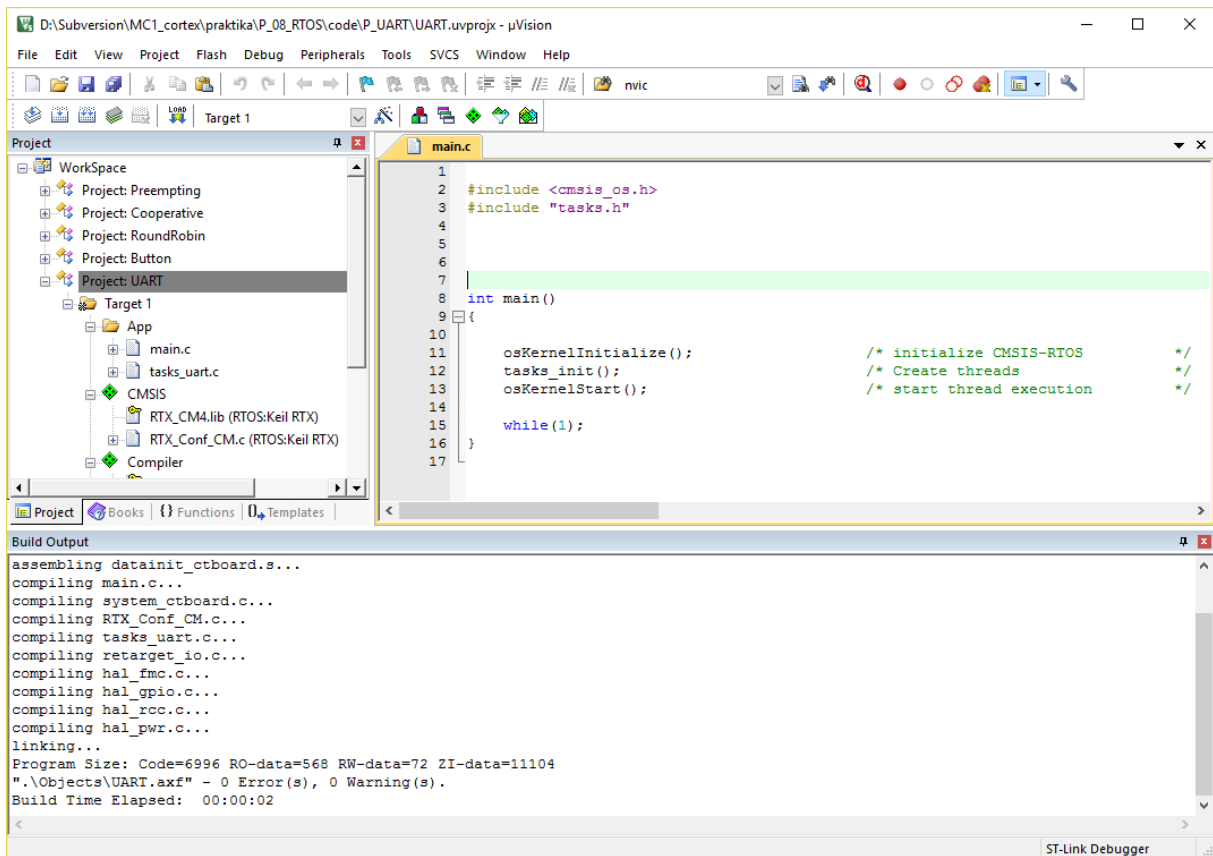


Abbildung 2: Keil Multiprojekt

Innerhalb eines Multiprojekts kann immer nur ein Projekt aktiv sein. Um das aktive Projekt zu wechseln, muss im Projektfenster, auf der linken Seite, das gewünschte Projekt markiert werden und mit einem Rechtsklick „Set as Active Project“ ausgewählt werden.

Die einzelnen Projekte sind komplett unabhängig, jedes Projekt muss separat eingestellt und konfiguriert werden.

- Öffnen Sie für dieses Praktikum das vorhandene Multiprojekt `.\code\Lab.uvmpw`.

4 Umgebung

Hardware

Verwenden Sie für **alle** Aufgaben in diesem Praktikum das Discovery Board ohne CT Board. Beim Discovery Board wurde zusätzlich SB9 verbunden damit die Trace Funktionalität (u.a. für den Event Viewer benötigt) genutzt werden kann (Siehe Abbildung 3).

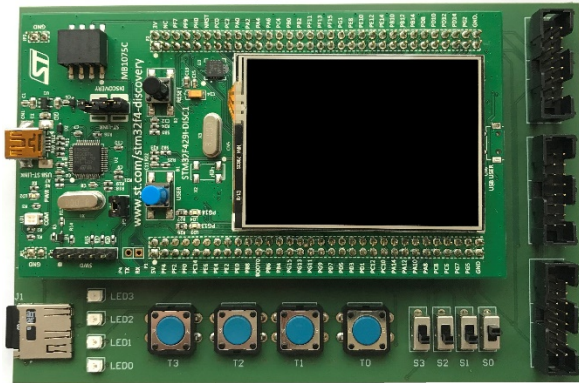


Abbildung 3: SWV enabled (SB9 verbunden) Discovery Board

RTOS Einstellungen

Die Einstellungen des Real Time Betriebssystems finden Sie hier:

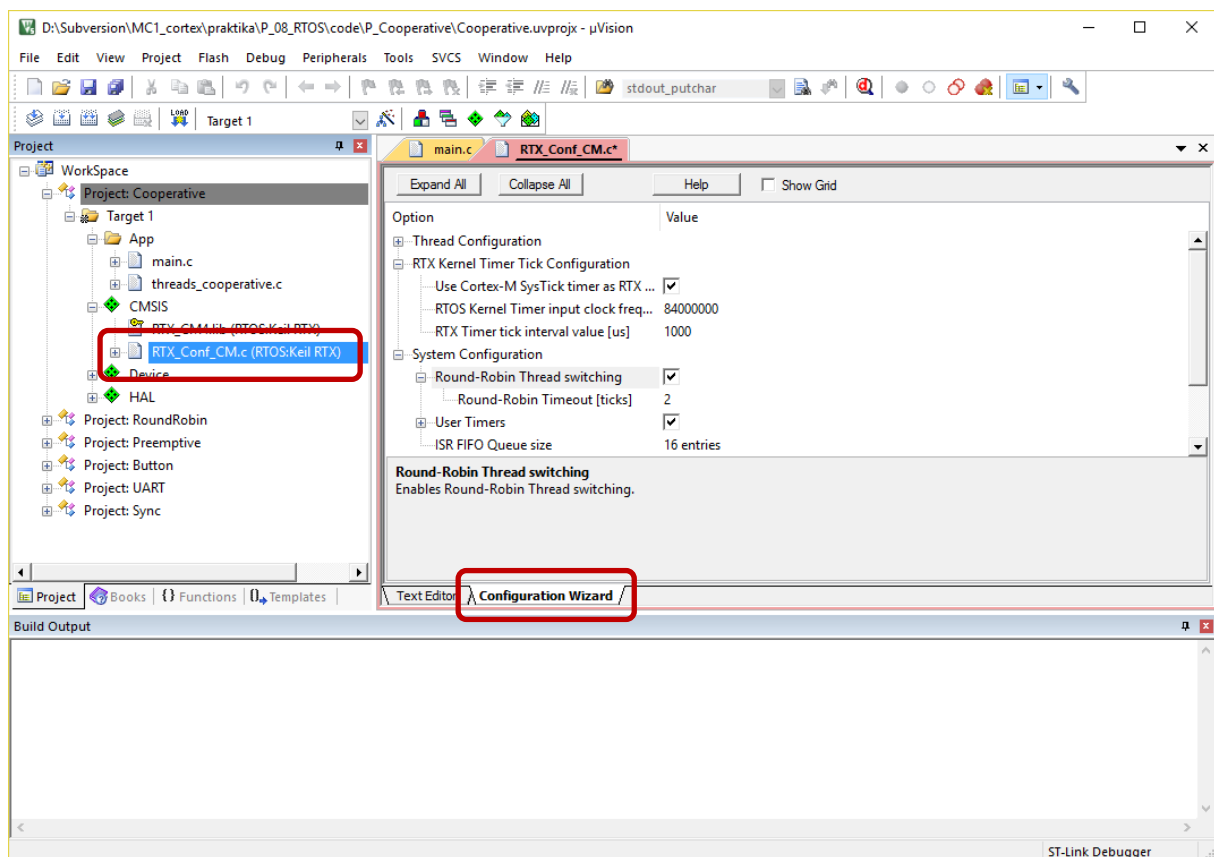


Abbildung 4: RTOS konfigurieren

Event Viewer

Das Scheduling-Verhalten können Sie im Keil Debugger mit dem Event Viewer untersuchen. (Debug -> OS Support -> Event Viewer) Weitere Informationen zum Event Viewer finden Sie unter http://www.keil.com/support/man/docs/uv4/uv4_db_dbg_event_viewer.htm.

Standardausgabe

Die Standardausgabe wird auf den UART1 Port gelegt (siehe Anhang 9). Mittels `printf(const char * format, ...)` können Sie beliebige Strings an den UART1 Port senden. Wie Sie den UART1 Port mit dem PC verbinden ist in Anhang 10 beschrieben.

5 Aufgabe – Scheduling

5.1 Cooperative

Bei Cooperative Scheduling gibt sich ein Thread selbst zur Arbitrierung frei ohne Timeout oder übergeordnete Priorität. Der Scheduler gibt dann dem nächsten Thread in der Queue die Erlaubnis zu starten. Beim Cooperative Scheduling muss Round Robin Scheduling ausgeschaltet sein und alle Threads müssen die gleiche Priorität haben.

Project	Cooperative
LED Grün (LD3)	GPIO Port G.13
LED Rot (LD4)	GPIO Port G.14

Zeigen Sie, dass Cooperative Scheduling vorliegt, indem Sie die Prioritäten der Threads ändern. Das Verhalten der Schaltung muss sich dann jeweils erklärbar ändern.

- Erstellen Sie in `threads_init()` zwei Threads, beide mit gleicher Priorität.
- Prüfen Sie für jeden der beiden Threads, ob das Erstellen erfolgreich war. Geben Sie entweder eine entsprechende Erfolgs- oder Misserfolgsmeldung auf der UART aus.
- Thread 1 schaltet die grüne LED ein, wartet ungefähr 0.5s, schaltet die LED wieder aus und wartet wieder ungefähr 0.5s. Danach gibt er sich frei (`yield`).
- Thread 2 macht dasselbe mit der roten LED.
- Verwenden Sie für das Warten die Funktion `wait_blocking(HALF_SECOND);`
- Experimentieren Sie mit den Prioritäten der Threads. Was stellen Sie fest?
- Fügen Sie im Anhang 11.1 einen Screenshot des Event Viewers mit Cooperative Scheduling und gleichen Prioritäten ein.

5.2 Round Robin Preemptive

Beim Round Robin Preemptive Scheduling erhält der laufende Thread eine konfigurierbare Anzahl Zeitschlitze zugeteilt. Ein Thread läuft so lange, bis der Thread blockiert wird, oder seine Zeitschlitze abgelaufen sind.

Project	Cooperative
---------	-------------

Untersuchen Sie die Unterschiede zwischen Cooperative und Round Robin Scheduling.

- Verwenden Sie dasselbe Projekt wie in Aufgabe 5.1.
- Schalten Sie Round Robin Scheduling ein.
- Fügen Sie im Anhang 11.2 einen Screenshot mit Round Robin Scheduling ein.
- Erklären Sie die Unterschiede

5.3 Messung Round Robin Preemptive Timeout

Das Round Robin Timeout lässt sich einstellen. Sie können einerseits das SysTick Intervall selbst (*RTX timer tick intervall [us]*, Siehe Abbildung 4) verändern sowie die Anzahl Ticks bis ein anderer Thread ausgeführt wird (*Round Robin timeout [ticks]*, Siehe Abbildung 4).

Project	RoundRobin
LED Grün (LD3)	GPIO Port G.13

Untersuchen Sie das Round Robin Timeout Verhalten mit Hilfe von zwei Threads. Messen Sie mit dem Oszilloskop das Round Robin Timeout. Verändern Sie das Round Robin Timeout und beweisen sie die Änderungen mit einer Messung.

- Erstellen Sie zwei Threads, beide mit gleicher Priorität.
- Thread 1 schaltet die grüne LED ein und wartet dann für ca. 0.5s.
- Thread 2 schaltet die grüne LED aus und wartet dann für ca. 0.5s.
- Verwenden Sie für das Warten folgende Funktion: `wait_blocking(HALF_SECOND)` ;
- Bestimmen Sie die Round Robin Timeouts anhand der SysTick Timer und Round Robin Timeout Einstellungen.
- Messen Sie mit dem Oszilloskop das Round Robin Timeout.
- Verändern Sie die Timeout Zeiten und verifizieren Sie damit Ihre Theorie.
- Fügen Sie im Anhang 11.3 einen Screenshot des Oszilloskops ein in dem die Timeout Zeit ersichtlich ist.

5.4 Preemptive

Beim Preemptive Scheduling kann ein Thread mit einer höheren Priorität einen Thread mit niedrigerer Priorität unterbrechen.

Project	Preemptive
LED Grün (LD3)	GPIO Port G.13
LED Rot (LD4)	GPIO Port G.14

Schreiben Sie Threads, mit welchen Sie das Preemptive Scheduling testen können. Zeigen Sie, dass Thread 1 erst aktiv wird, wenn Thread 2 im Zustand *waiting* ist.

- Erstellen Sie zwei Threads, Thread 2 soll eine höhere Priorität haben.
- Thread 1 schaltet die grüne LED ein, wartet ungefähr 0.5s, schaltet die LED wieder aus und wartet wieder ungefähr 0.5s (`wait_blocking(HALF_SECOND)`).
- Thread 2 macht dasselbe mit der roten LED.
- Fügen Sie am Ende des while-Loop von Thread 2 ein `osDelay(40u)` ein, damit der höher priorisierte Thread 2 für 40ms schlafen gelegt wird.
- Fügen Sie im Anhang 11.4 einen Screenshot des Event Viewers ein.

6 Aufgabe – Signals

Signals werden verwendet um einem anderen Thread zu signalisieren, dass ein bestimmter Event aufgetreten ist. Threads können auf Signals warten, welche auslösen oder löschen.

Project	Button
User Button	GPIO Port A.0
LED Grün (LD3)	GPIO Port G.13

Auf dem Discovery Board befindet sich ein User Button. Dieser Button soll mit Hilfe von Signals die grüne LED ein- bzw. ausschalten.

- Erstellen Sie zwei Threads, beide mit gleicher Priorität.
- Thread 1 pollt den User Button und setzt, falls er gedrückt wurde, ein Signal ab. Implementieren Sie eine geeignete Flankendetektion.
- Thread 2 wartet auf das Signal von Thread 1 und schaltet, wenn es empfangen wurde, die grüne LED ein bzw. aus.

7 Aufgabe – Mutexes

Wenn mehrere Threads auf eine gemeinsame Ressource, z.B. eine Variable, zugreifen möchten, kann es zu Problemen kommen; Der Zugriff auf eine Ressource ist selten atomar (Variable lesen, verändern und wieder zurückschreiben). Damit es nicht zu Inkonsistenzen kommt, z.B. aufgrund eines Thread Switches, muss der Zugriff auf diese Ressource gesteuert werden. Dies geschieht mit Hilfe von Mutexes. Ein Mutex wird vor dem eigentlichen Zugriff angefordert und danach wieder freigegeben. Ein anderer Thread, der ebenfalls auf diese Ressource zugreifen möchte, wird blockiert. Somit ist sichergestellt, dass ein Thread den Zugriff vollständig abschliessen kann.

Project	UART
---------	------

Setzen Sie das Programm zuerst ohne Mutex um, damit Sie den Effekt sehen. Lösen Sie das Problem nachher mit einem Mutex.

- Erstellen Sie zwei Threads, beide mit gleicher Priorität.
- Beide Threads sollen jeweils die vorgegebene statische Variable `count` hochzählen und diese mittels `printf()` ausgeben.
- Verwenden Sie nach der Ausgabe die Funktion `wait_blocking(HALF_SECOND)` um die Ausgabe am Bildschirm verfolgen zu können.
- Mit `printf()` können Sie die Ausgabe noch formatieren. Weitere Informationen finden Sie unter https://www.tutorialspoint.com/c_standard_library/c_function_printf.htm.
- Für einen Zeilenumbruch müssen Sie `\n\r` verwenden, statt nur `\n`.
- Sorgen Sie dafür, dass bei jeder Ausgabe ersichtlich ist, von welchem Thread sie stammt.
- Zeigen Sie beide Versionen, mit und ohne Mutex.

8 Bewertung

Das Praktikum wird mit maximal 3 Punkten bewertet:

- | | |
|------------------------|---------|
| • Aufgabe – Scheduling | 1 Punkt |
| • Aufgabe – Signals | 1 Punkt |
| • Aufgabe – Mutexes | 1 Punkt |

Punkte werden nur gutgeschrieben, wenn die folgenden Bedingungen erfüllt sind:

- Der Code muss sauber strukturiert und kommentiert sein.
- Das Programm ist softwaretechnisch sauber aufgebaut.
- Die Funktion des Programmes wird erfolgreich vorgeführt.
- Der/die Studierende muss den Code erklären und zugehörige Fragen beantworten können.
- Der Zusatzpunkt wird nur vergeben, wenn alle anderen Aufgaben gelöst sind.

9 Anhang – STDOUT über UART

Um die Standardausgabe auf den UART1 Port zu legen, öffnen Sie *Project* → *Manage* → *Runtime Environment*.... Aktivieren Sie *Compiler* → *I/O* → *STDOUT* und wählen Sie im dazugehörigen Dropdown Feld *USER* aus (Siehe Abbildung 5).

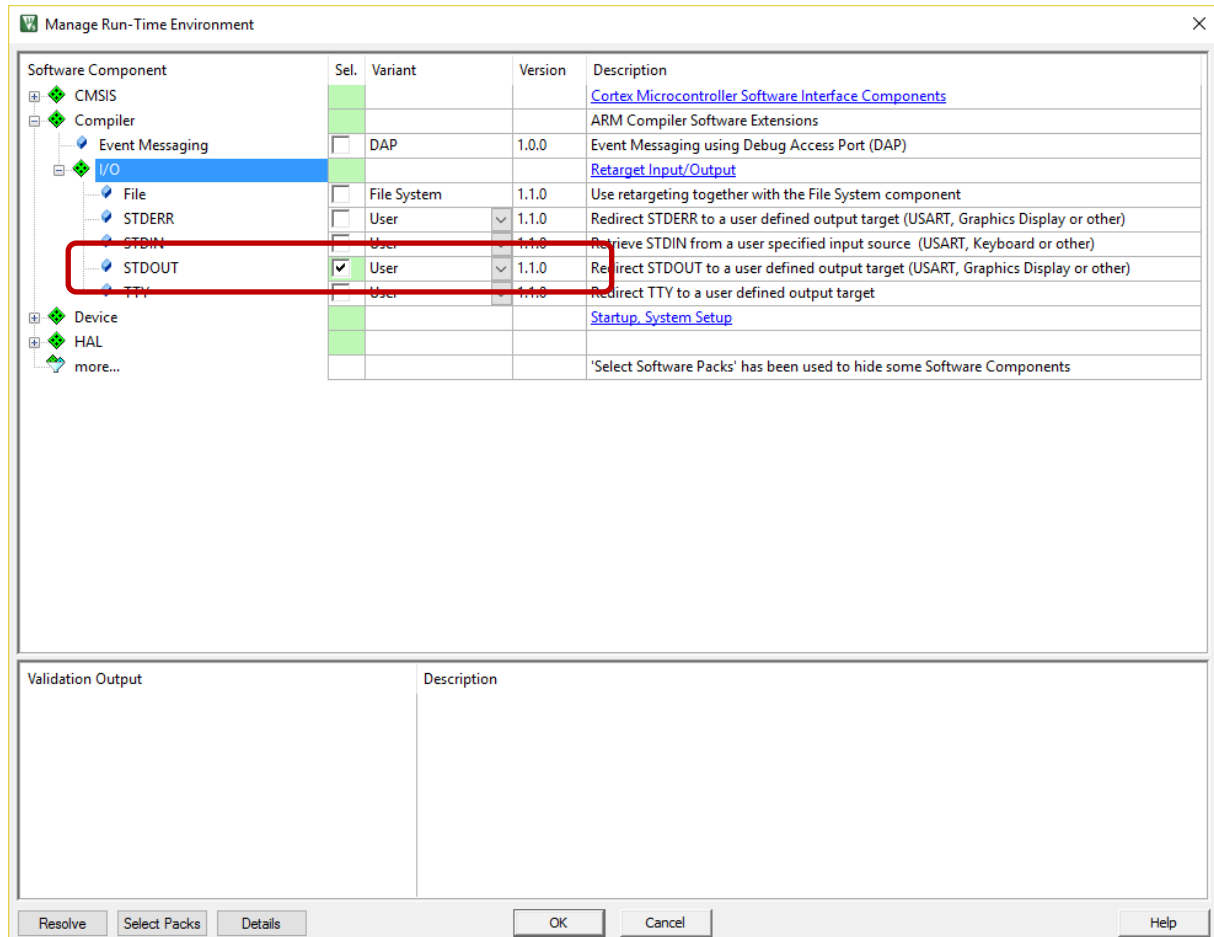


Abbildung 5: STDOUT aktivieren

Anschliessend müssen Sie den entsprechenden UART konfigurieren und eine spezielle Funktion `int stdout_putchar(int ch)` erstellen in welcher jeweils ein Zeichen ausgegeben wird. In diesem Projekt sind aber beide Punkte im Modul *uart.c* bereits implementiert.

10 Anhang – UART an PC anschliessen

Beim Discovery Version 1 (DISC1) ist UART1 über den ST-Link Debugger mit dem PC verbunden. Verwenden Sie den ST-Link Virtual COM Port (im Gerätemanager suchen, siehe Abbildung 6).

Öffnen Sie mit PuTTY eine Verbindung mit 115'200 bps über den entsprechenden COM Port.

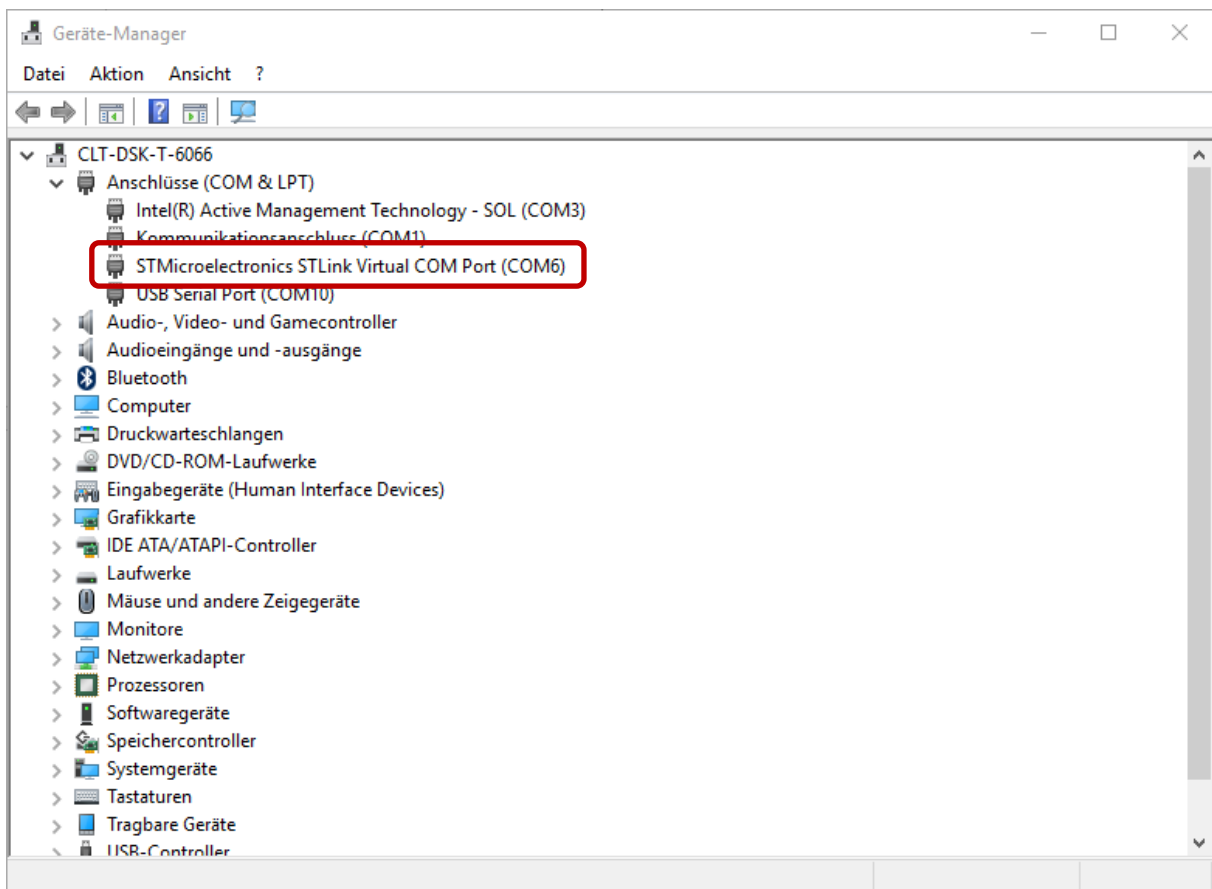


Abbildung 6: Geräte-Manager

11 Anhang – Screenshots zu Aufgaben

11.1 Cooperative



Abbildung 7: Cooperative Scheduling

11.2 Round Robin Preemptive



Abbildung 8: Round Robin Scheduling

11.3 Messung Round Robin Preemptive Timeout

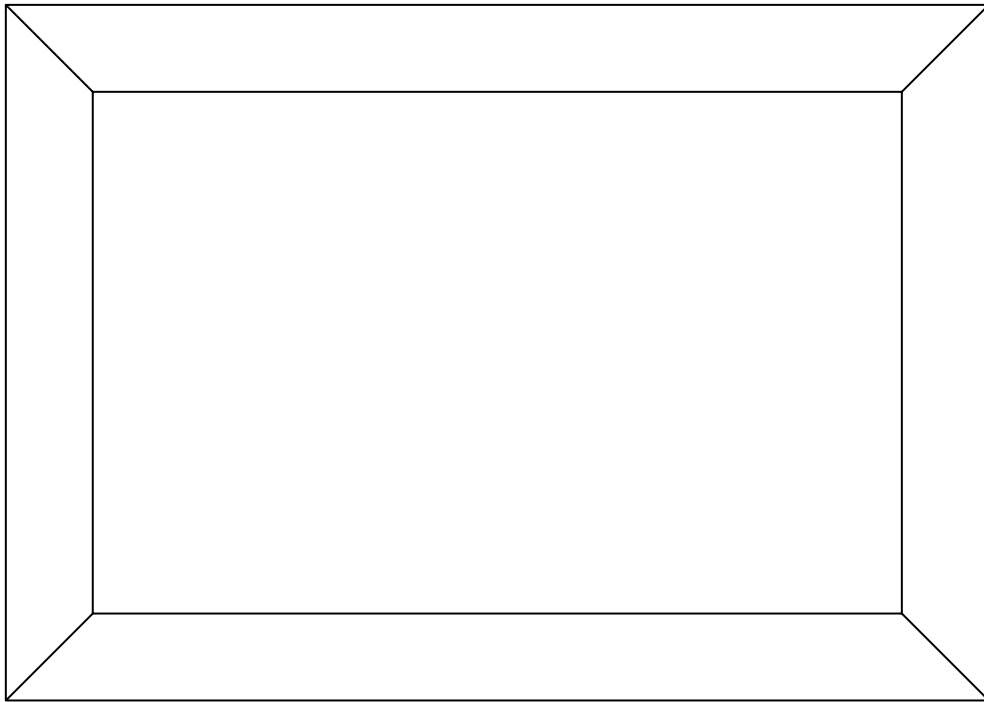


Abbildung 9: Output grüne LED (Port G.13)

11.4 Preemptive

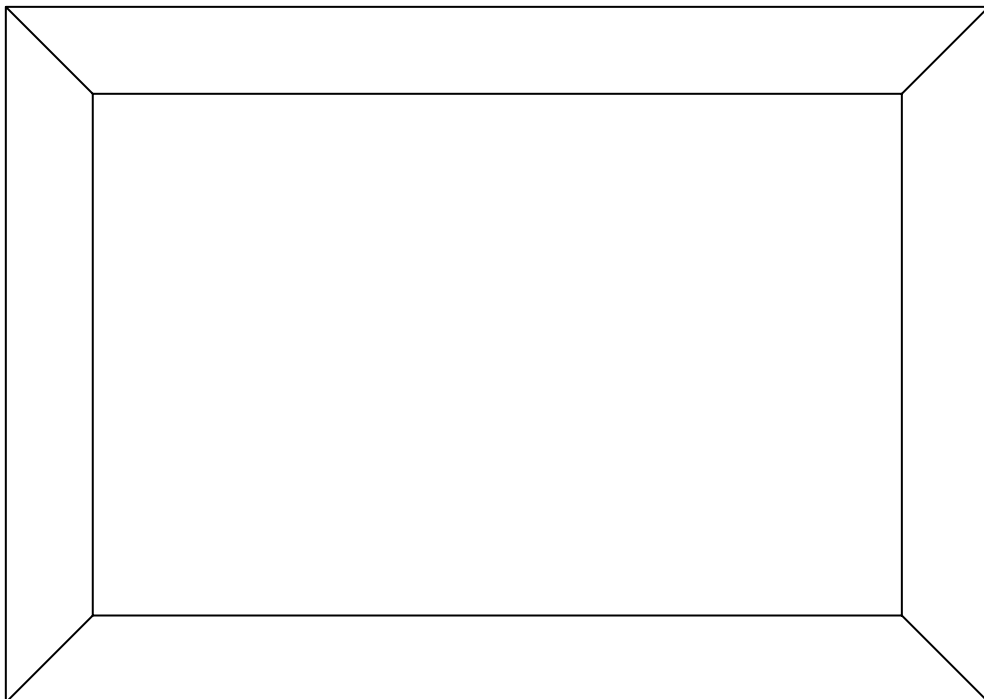


Abbildung 10: Preemptive Scheduling