WILHELM BÜCHNER HOCHSCHULE

Studiengang:
Embedded Systems and Digital Technologies

B-Aufgabe

B-EBF01-XX6-K02

Embedded Software Engineering

Student: Kilian Vogler
kilian.vogler@gmail.com

Matrikelnummer: 861105

Abgabedatum: 25.09.2025

Inhaltsverzeichnis

Aufgabenstellung B-EBF01-XX6

text

text

text

1 Zustandsdiagramm Scheibenwischersteuerung

Aufgabenstellung

Betrachten Sie die Steuerung durch Software eines Scheibenwischers. Der Scheibenwischer kann ausgeschaltet sein, oder auf der Stufe „einmal wischen“, Stufe 1 oder Stufe 2 sein. Wenn die Option „Einmal wischen“ ausgewählt wird, wischt der Scheibenwischer nur einmal und schaltet sich wieder aus. Der Fahrer hat die Möglichkeit ein automatisches Wischen der Windschutzscheiben auszuwählen, wenn es regnet. Falls der Scheibenwischer ausgeschaltet ist, wird dieses automatische System aktiviert, sobald Regen auf die Windschutzscheibe fällt. Erstellen Sie ein Zustandsdiagramm (state diagram) für den Scheibenwischer.

Die grundlegenden Funktionen des Scheibenwischers sind in die folgenden Zustände unterteilt:

- Aus (OFF)
- Einmal wischen (WIPE_ONCE)
- Stufe 1 (LEVEL_1)
- Stufe 2 (LEVEL_2)
- Automatisch (AUTO)

Das folgende Zustandsdiagramm (ZD) (engl. State Machine Diagram (SMD)) modelliert die Zustände (Abbildung 1) und die möglichen Übergänge zwischen ihnen. Die mit „Driver“ gekennzeichneten Übergänge zeigen an, dass der Fahrer manuell eine Änderung des Wischerzustands vornimmt. Wohingegen die mit „Rain sensor“ gekennzeichneten Übergänge anzeigen, dass die Änderung des Wischerzustands automatisch durch das System erfolgt, basierend auf dem Regensensor.

Das Diagramm ist in voller Größe in Abbildung 1 auf der nächsten Seite dargestellt.

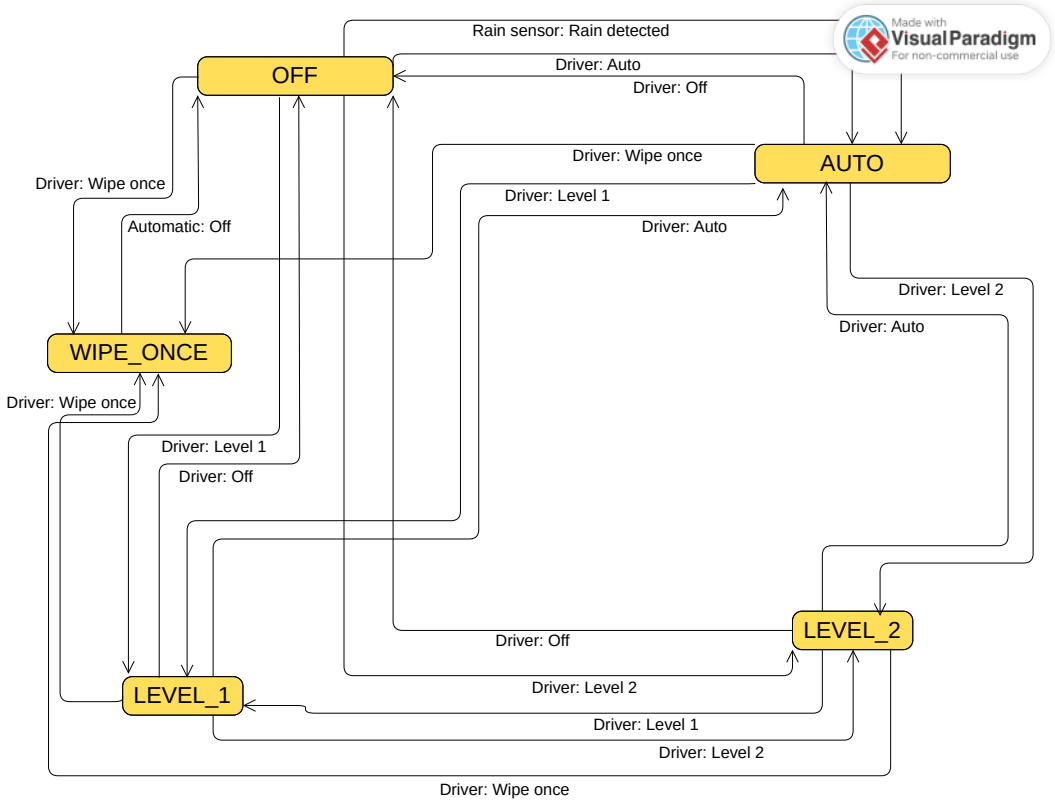

Abbildung 1: ZD Scheibenwischer (eigene Darstellung mit [1])

2 Akteure & Use-Case-Diagramm der Scheibenwischersoftware

2.1 Identifikation der Akteure

Aufgabenstellung

Was sind Akteure und wie interagieren sie mit dem zu entwickelnden System?

Für das in Abschnitt 1 betrachtete Scheibenwischersystem lassen sich die folgenden Akteure und deren Interaktionen mit dem System identifizieren:

Akteure:

- **Fahrer / „Driver“ (primärer Akteur):**
  wählt „Aus“, „Einmal“, „Stufe 1“, „Stufe 2“ oder „Auto“.
- **Regensensor / „Rain sensor“ (sekundärer Akteur):**
  liefert Information „Regen erkannt / kein Regen“ sowie die „Intensität des Regens“.

Interaktion:

- Der Fahrer löst durch Eingaben Zustandswechsel des Wischers aus.
- Der Sensor interagiert automatisch und triggert Zustandswechsel im Automatikmodus und passt die Wischgeschwindigkeit an die Regenintensität an.
- Das System reagiert, indem es die Wischermotoren steuert und die gewünschte Funktion ausführt. Sowie schaltet sich bei „Einmal wischen“ nach einem Zyklus wieder aus.

2.2 Use-Case-Diagramm: Scheibenwischersoftware

Aufgabenstellung

Erstellen Sie ein Anwendungsfalldiagramm (use-case diagramm) der Steuerungssoftware des Scheibenwischers in dieser Form (nur als Beispiel, das nichts über die Zahl der Use Cases und Akteure im gegebenen Fall aussagt): Der Regensensor sollte als Akteur modelliert werden.

Im folgenden Anwendungsfalldiagramm (engl. Use Case Diagram (UCD)) sind die Akteure Fahrer („Driver“) und Regensensor („Rain sensor“) sowie deren Interaktionen mit dem Scheibenwischersystem (Windshield wiper software) in Abbildung 2 dargestellt.

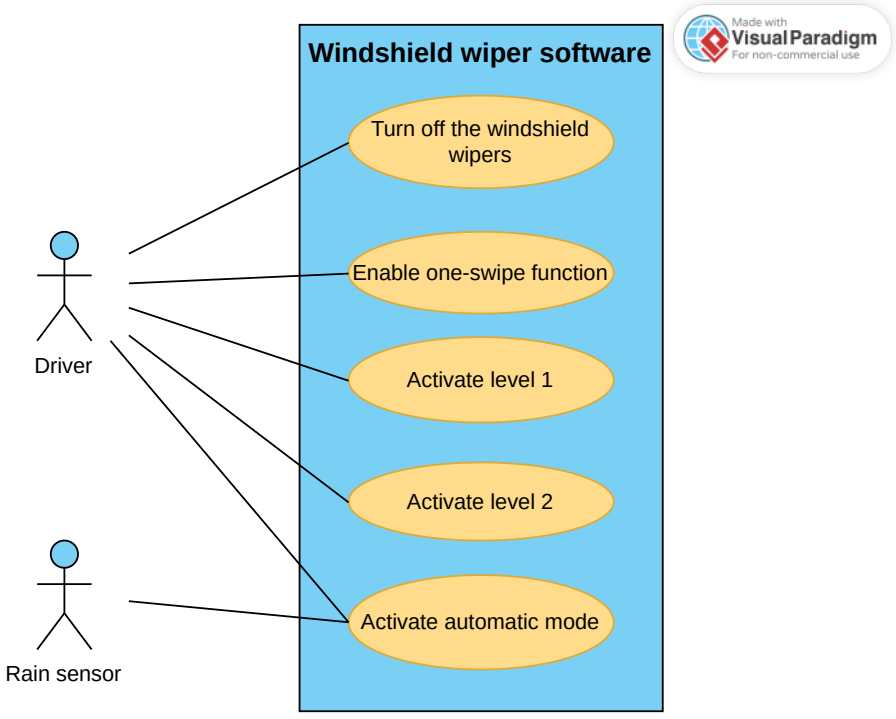

Abbildung 2: UCD Scheibenwischer (eigene Darstellung mit [1])

3 Aktivitätsdiagramm: Automatischer Scheibenwischerbetrieb

Aufgabenstellung

Erstellen Sie ein UML-Aktivitätsdiagramm (Activity Diagram) für den Scheibenwischer für das folgende Szenario: Der Scheibenwischer wird von Motoren angetrieben und seine Geschwindigkeit über einen Regensensor reguliert. Nehmen Sie an, dass das automatische Wischen schon eingeschaltet ist.

Im UML-Aktivitätsdiagramm (engl. Activity Diagram (AD)) in Abbildung 3 ist der Ablauf des automatischen Scheibenwischerbetriebs modelliert. Die Intensität des Regens ist in der Darstellung in die Zustände „kein Regen“, „leichter Regen“, „mäßiger Regen“ und „starker Regen“ unterteilt. Es wurde auch die manuelle Abschaltung des Scheibenwischers berücksichtigt. Nach einem Zeitintervall wird der Regensensor erneut abgefragt, um die Wischgeschwindigkeit entsprechend der aktuellen Regenintensität anzupassen.

Das UML-Aktivitätsdiagramm ist in voller Größe in Abbildung 3 auf der nächsten Seite dargestellt.

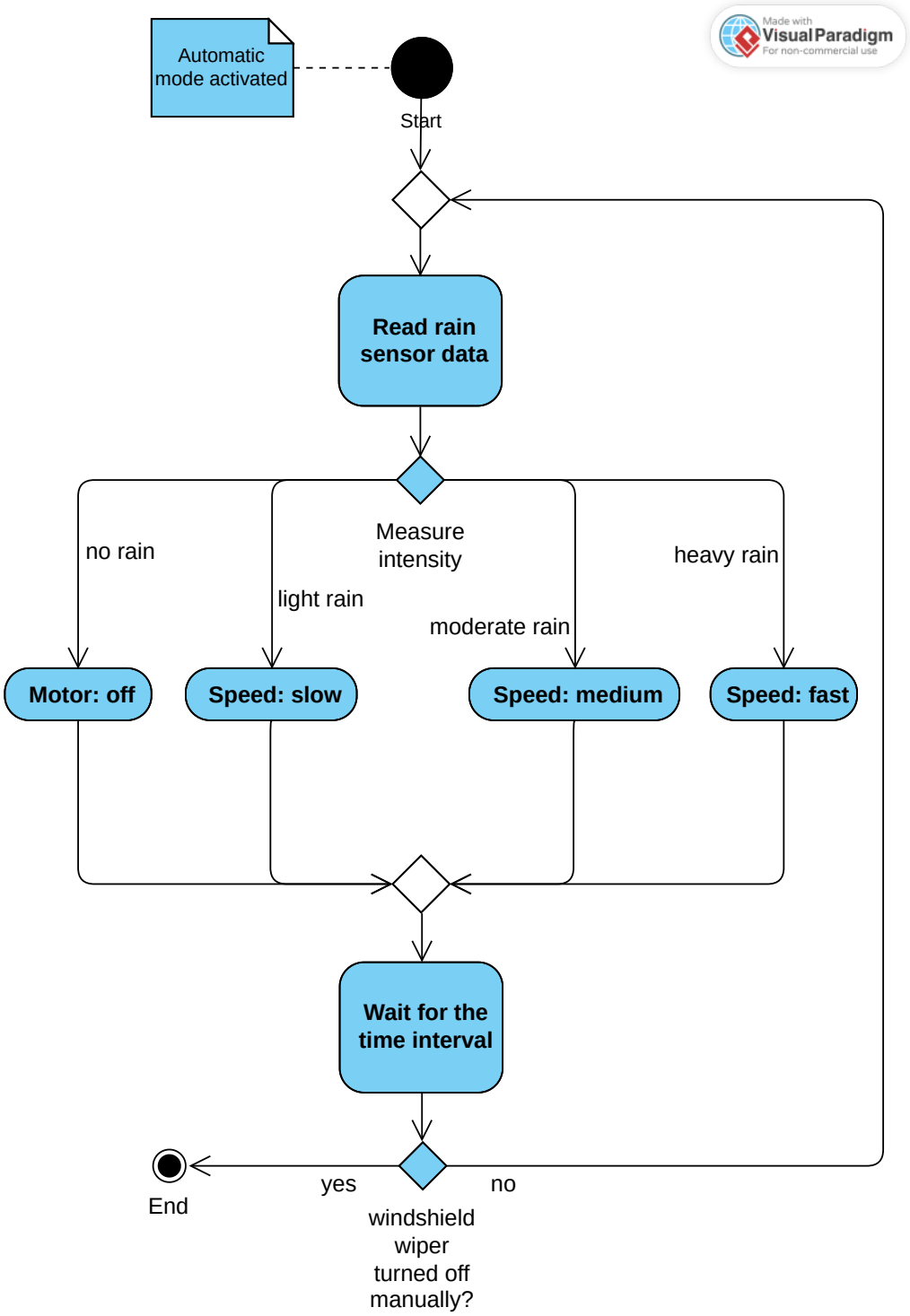

Abbildung 3: UML-AD Scheibenwischer (eigene Darstellung mit [1])

4 Nebenläufigkeit eingebetteter Systeme

4.1 Inhärente Nebenläufigkeit eingebetteter Systeme

Aufgabenstellung

Erläutern Sie, warum eingebettete Systeme inhärent nebenläufig sind und warum dies einen Grund für ihre hohe Entwurfskomplexität darstellt.

Eingebettete Systeme sind inhärent nebenläufig, da sie simultan auf verschiedene asynchrone Ereignisse reagieren müssen. Dies zeigt sich im Scheibenwischersystem: Während das System kontinuierlich Regensensordaten auswertet und die Motorgeschwindigkeit entsprechend anpasst, muss es gleichzeitig auf manuelle Benutzereingaben reagieren können. Der Regensensor arbeitet in regelmäßigen Intervallen, unabhängig von der Motorsteuerung oder Benutzereingaben. Hinzu kommt eine interrupt gesteuerte Architektur, bei der externe Events wie Sensormessungen oder Timer-Interrupts jederzeit eintreten und die Hauptausführung unterbrechen können.

Die hohe Entwurfskomplexität entsteht durch Race Conditions beim gleichzeitigen Zugriff auf gemeinsame Systemzustände und die Notwendigkeit deterministischer Echtzeitreaktionen trotz paralleler Prozesse und die schwierige Vorhersagbarkeit des Systemverhaltens. Darüber hinaus erfordert die Beschränkung auf begrenzte Ressourcen, wie CPU-Zeit, Speicher und Energie, eine sorgfältige Priorisierung und Synchronisation der Tasks. Debugging wird zudem erschwert, da timing abhängige Fehler oft nicht reproduzierbar sind und spezielle Analysetools benötigt werden, um Race Conditions oder Deadlocks aufzuspüren [2, 3].

4.2 POSIX Threads: Umsetzung der Nebenläufigkeit

Aufgabenstellung

Nebenläufige Tasks werden in der Programmiersprache C mithilfe von POSIX Threads implementiert. Erklären Sie kurz, wie Threads die Nebenläufigkeit implementieren.

POSIX Threads implementieren Nebenläufigkeit durch die Aufteilung der Programmlogik in mehrere parallel ausführbare Einheiten. Jeder Thread erhält einen eigenen Ausführungskontext: Stack, Programmzähler, Register, während alle Threads denselben Adressraum teilen. Das ermöglicht effizienten Datenaustausch über gemeinsame Variablen [4].

Das Betriebssystem verwendet präemptives Scheduling, um Threads zeitscheibenbasiert oder prioritätsgesteuert auf verfügbare CPU-Kerne zu verteilen. Moderne pthread-Implementierungen nutzen Kernel-Level-Threads für echte Parallelität auf Mehrkernsystemen. Synchronisationsmechanismen wie Mutexes und Condition Variables koordinieren den Zugriff auf gemeinsame Ressourcen.

4.3 Zusätzliche Thread-Funktionen im Scheibenwischersystem

Aufgabenstellung

Betrachten Sie wieder das Scheibenwischersystem. Das Folgende zeigt eine Funktion (im Pseudocode), die ein Thread ausführen kann. Geben Sie zwei weitere Funktionen an, die beim Scheibenwischersystem als Threads ausgeführt werden können.

Im Folgenden zwei Threads für das Scheibenwischersystem. Zum einen der Regensensor-Thread und der Motorsteuerungs-Thread.
Mit dem Regensensor-Thread wird die Regenintensität kontinuierlich überwacht und die Wischgeschwindigkeit entsprechend angepasst.

```
1  void* RainSensor_Thread() {
2      while(1) {
3          int rainIntensity = readRainSensor();
4
5          int wiperSpeed;
6          if(rainIntensity == 0) {
7              wiperSpeed = OFF;
8          }
9          else if(rainIntensity <= LIGHT) {
10             wiperSpeed = SLOW;
11         }
12         else if(rainIntensity <= MODERATE) {
13             wiperSpeed = MEDIUM;
14         }
15         else {
16             wiperSpeed = FAST;
17         }
18
19         writeSpeedToBuffer(wiperSpeed);
20
21         waitSamplingInterval();
22
23         if(isSystemManuallyOff()) break;
24     }
25     return NULL;
26 }
```

Quellcode 1: Regensensor-Thread

Mit dem Motorsteuerungs-Thread wird die Wischgeschwindigkeit basierend auf den Daten des Regensensors eingestellt und der Wischzyklus gestartet. Es wird zudem auch nach dem Wischzyklus überprüft, ob das System manuell ausgeschaltet wurde.

```
void* MotorControl_Thread() {
    while(1) {
        int currentSpeed = readSpeedFromBuffer();

        switch(currentSpeed) {
            case OFF:
                stopMotor();
                break;
            case SLOW:
                setMotorSpeed(SLOW_VALUE);
                startWipeCycle();
                break;
            case MEDIUM:
                setMotorSpeed(MEDIUM_VALUE);
                startWipeCycle();
                break;
            case FAST:
                setMotorSpeed(FAST_VALUE);
                startWipeCycle();
                break;
        }

        if(isSystemManuallyOff()) {
            stopMotor();
            break;
        }

        usleep(MOTOR_CYCLE_DELAY);
    }
    return NULL;
}
```

Quellcode 2: Motorsteuerungs-Thread

4.4 Kommunikation und Synchronisation der Threads

Aufgabenstellung

Beschreiben Sie, warum die Threads miteinander kommunizieren oder sich synchronisieren müssen. Geben Sie auch an, wie Sie die Kommunikation oder Synchronisation implementieren würden (nur das Mittel, nicht den Code).

Die Threads müssen miteinander kommunizieren, weil sie gemeinsame Daten austauschen und koordiniert arbeiten müssen. Der Regensensor-Thread bestimmt basierend auf Sensormessungen die erforderliche Motorgeschwindigkeit, die der Motorsteuerungs-Thread umsetzen muss. Ohne Kommunikation könnte der Motor nicht auf Änderungen der Regenintensität reagieren.

Die Synchronisation ist erforderlich, um Race Conditions zu vermeiden. Wenn beide Threads gleichzeitig auf den gemeinsamen Geschwindigkeitspuffer zugreifen, während der Regensensor-Thread eine neue Geschwindigkeit schreibt und der Motorsteuerungs-Thread diese liest, können inkonsistente Daten entstehen. Dies könnte zu unvorhersagbarem Motorverhalten oder Systemfehlern führen. Um dies zu verhindern, gibt es die folgenden Mechanismen in der Implementierung:

- **Mutex (pthread_mutex_t)**: Schützt den kritischen Abschnitt beim Zugriff auf den gemeinsamen Geschwindigkeitspuffer und verhindert gleichzeitige Lese-/Schreibzugriffe.
- **Condition Variables (pthread_cond_t)**: Signalisieren Ereignisse zwischen den Threads, z. B. wenn der Regensensor-Thread eine neue Geschwindigkeit ermittelt hat oder wenn das System manuell ausgeschaltet wird.
- **Shared Buffer**: Ein geschützter Speicherbereich zur Übertragung der Geschwindigkeitswerte zwischen den Threads.
- **Atomic Variables**: Für einfache Statusflags wie den manuellen Ausschaltbefehl, um lockfreie Synchronisation zu ermöglichen.

Diese Mechanismen gewährleisten Datenintegrität trotz paralleler bzw. überlappender Ausführung der Threads.

5 Echtzeitsysteme & Verkehrszeichenerkennung

5.1 Echtzeitsystem-Analyse: Scheibenwischer

Aufgabenstellung

Ist der Scheibenwischer in Aufgabe 1 ein Echtzeitsystem? Begründen Sie Ihre Antwort?

Der Scheibenwischer aus Abschnitt 1 ist ein Echtzeitsystem, weil er zeitkritisch auf äußere Ereignisse, wie hier das Fallen von Regen, reagieren muss. Die Steuerungssoftware darf die Aktivierung des Wischers bei Regen nicht zu stark verzögern, dass die Sicht des Fahrers zu sehr beeinträchtigt wird. Hier kann eine späte Reaktion zu Gefährdungen führen und es müssen alle Rechen- und Sensorzugriffe innerhalb definierter Fristen abgeschlossen sein. Da jedoch gelegentliche geringfügige Verzögerungen um wenige Millisekunden nicht unmittelbar katastrophale Folgen haben, handelt es sich um ein weiches Echtzeitsystem [5].

5.2 Beispielhafte Anwendungen der Verkehrszeichenerkennung

Aufgabenstellung

Beschreiben Sie mithilfe von Beispielen, wie ein Verkehrszeichenerkennungssystem verwendet werden könnte.

Verkehrszeichenerkennungssysteme (Traffic Sign Recognition (TSR)) analysieren in Echtzeit Kamerabilder, um Verkehrszeichen zu identifizieren und deren Information an den Fahrer oder Fahrassistenzsysteme zu übermitteln. Beispiele hierfür sind die folgenden Systeme:

- Anzeige der zulässigen Höchstgeschwindigkeit im Kombiinstrument oder Head-Up-Display, sowie Warnung bei Überschreitung.
- Automatische Anpassung der Geschwindigkeit im adaptiven Tempomaten basierend auf erkannten Geschwindigkeitsbegrenzungen.

5.3 Echtzeitfähigkeit und Klassifizierung von Verkehrszeichenerkennungssystemen

Aufgabenstellung

Erläutern Sie, warum Verkehrszeichenerkennungssysteme echtzeitfähig sein müssen. Handelt es sich um harte Echtzeitsysteme? Begründen Sie Ihre Antwort.

Verkehrszeichenerkennung muss echtzeitfähig sein, weil erkannte Informationen nur innerhalb eines engen Zeitfensters relevant sind. Wird ein Tempolimit zu spät erkannt, kann das Assistenzsystem nicht rechtzeitig die Fahrzeuggeschwindigkeit anpassen und der Fahrer könnte in eine Gefahrenlage geraten bzw. an der Stelle mit Beginn der Geschwindigkeitsbegrenzung noch eine zu hohe Geschwindigkeit fahren.

Da in diesem Beispiel eine einzelne Fristverletzung typischerweise nicht sofort katastrophal ist, es könnte ein verpasstes Schild zu Komforteinbußen oder Bußgeldrisiken führen, wird TSR üblicherweise als weiches Echtzeitsystem klassifiziert. Harte Echtzeitsysteme erfordern, dass jede Deadline ohne Ausnahme absolut eingehalten wird, was hier nicht in gleicher Weise gefordert ist [5]. Ein Beispiel für ein hartes Echtzeitsystem im Bereich der Fahrzeugtechnik wäre ein Airbag-Steuergerät, bei dem eine Fristverletzung zu lebensbedrohlichen Folgen führen kann.

Literaturverzeichnis

[1] *Visual Paradigm - Online-Produktivitätssuite*. https://online.visual-paradigm.com/de/. (Besucht am 31. 08. 2025) (zitiert auf den Seiten 5, 7, 9).

[2] D. M. Cummings. *Managing Concurrency in Complex Embedded Systems*. https://www.state-machine.com/doc/Cummings1006.pdf. (Besucht am 21. 09. 2025) (zitiert auf Seite 10).

[3] *Concurrency and Interrupts in Microcontrollers and Embedded Systems - Technical Articles*. https://www.allaboutcircuits.com/technical-articles/introduction-to-concurrency-interrupts-microcontrollers-embedded-systems/. (Besucht am 21. 09. 2025) (zitiert auf Seite 10).

[4] *Linux Tutorial: POSIX Threads*. https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html. (Besucht am 22. 09. 2025) (zitiert auf Seite 11).

[5] *How to Achieve Deterministic Behavior in Real-Time Embedded Systems - Lance Harvie*. https://www.embeddedrelated.com/showarticle/1742.php. (Besucht am 23. 09. 2025) (zitiert auf den Seiten 15, 16).

Abkürzungsverzeichnis

AD Activity Diagram . . . . . 8, 9, II

SMD State Machine Diagram 4

TSR Traffic Sign Recognition 15, 16

UCD Use Case Diagram . . . . . 7, II

ZD Zustandsdiagramm . . . . . 4, 5, II

Abbildungsverzeichnis

Quellcodeverzeichnis