Technische Hochschule Ingolstadt

Fakultät Informatik

Studiengang Flug- und Fahrzeuginformatik

Bachelorarbeit

Analyse und Implementierung von Watchdog-Mechanismen in sicherheitskritischen Systemen

Vorgelegt von

Khalil Ibesh

Wissenschaftliche Abschlussarbeit
zur Erlangung des akademischen Grades

B. Sc.

Erstprüfer	Prof. Dr. Robert Gold
Zweitprüfer	Prof. Dr. Andreas Frey
Ausgabedatum	06. Juni 2023
Abgabedatum	10. November 2023

Erklärung

Ich erkläre hiermit, dass ich die Arbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Ingolstadt, 24. Oktober 2023

…….

Khalil Ibesh

Abkürzungen

ACP	Advanced Communication Processors
ASIL	Automotive Safety Integrity Level
AUTOSAR	Automotive Open System Architecture
DIO	Digital Input/Output
GPT	General Purpose Timer
I2C	Inter-Integrated Circuit
IWDG	Internal watch dog
NMI	Non-Maskable Interrupt
Q&A	Questions and Answers
RTOS	Real-Time Operating System
SPI	Serial Peripheral Interface
WDT	Watchdog Timer
WWDG	Window Watchdog Timer

Abbildungsverzeichnis

Abbildung 1: Eine typische Watchdog-Konfiguration [10] 8
Abbildung 2: Diese Schaltung Löst bei Zeitüberschreitung einen kontinuierlichen Reset aus [9] 10
Abbildung 3: Diese Schaltung erzeugt Reset-Impulse bei Zeitüberschreitung [9] 10
Abbildung 4:Ein flexibles, Watchdog-gesteuertes, ausfallsicheres System [9] 11
Abbildung 5: Zweistufiger Watchdog [9] 12
Abbildung 6: Dreistufiger Watchdog [9] 13
Abbildung 7: Überwachungsmechanismus mit Watchdog-Flow-Monitoring in der geschichteten AUTO-SAR-Architektur [3] 15
Abbildung 8:Überwachungsmechanismus mit Watchdog-Flow-Monitoring in der geschichteten Microsar-Architecture [3] 16
Abbildung 9: AUTOSAR Watchdog Stack [12] 18
Abbildung 10: Triggern des Watchdogs [13] 21
Abbildung 11: Message Queue [8] 23

Listingverzeichnis

Listing 1: Kicking the dog [10] 9
Listing 2: Eine Nachricht erstellen 23
Listing 3: Eine Nachrichtenwarteschlange entfernen 24
Listing 4 Eine Nachricht senden 24
Listing 5: Eine Nachricht empfangen 24
Listing 6: Erzeugung Watchdog- und Controller-Prozesse 25
Listing 7: Implementierung des Controller-Prozesses 27
Listing 8 :Implementierung der Timeout-Fehlerfunktion 28
Listing 9: Erzeugung eines Deadlocks 29
Listing 10: Implementierung der Temperaturfehler-Funktion 30
Listing 11: Implementierung des Watchdog-Prozesses 32
Listing 12: Implementierung der Fehlerbehebungsfunktion 33
Listing 13: Wiederherstellung des sicheren Zustands 34
Listing 14: Implementierung der Aktivierung des Kühlsystems 35
Listing 15:Ausgabe des Deadlock-Szenarios 36
Listing 16: Ausgabe des Timeout-Szenarios 37
Listing 17: Ausgabe des Temperatur-Szenarios 38
Listing 18 Kurzzeitige Ausgabe während Programm-Ausführung……..……..40
Listing 19: Ausgabe einer früheren Nachricht...……..41

Inhaltsverzeichnis

1 Einleitung 1
1.1 Hintergrund und Bedeutung 1
1.2 Zielsetzung der Arbeit 2
2 Theoretischer Hintergrund 3
2.1 Grundlagen von Watchdog 3
2.1.1 Definition und Zweck von Watchdogs 3
2.1.2 Arten von Watchdogs 4
2.1.3 Funktionen des Watchdogs 6
2.1.4 Konzept und Architektur eines Watchdogs 8
2.2 ISO 26262 und AUTOSAR 13
2.2.1 Features zur Health-Monitoring für Watchdogs 16
2.2.2 Watchdog-Stack in AUTOSAR 18
3 Methodik und Implementierung 22
3.1 Entwicklungsumgebung 22
3.2 Auswahl der Kommunikationsmethode 22
3.3 Erzeugung der zwei Prozesse 25
4 Implementierung des Steuerprozesses 27
4.1 Beschreibung des Steuerprozesses 27
4.1.1 Umsetzung der Timeout-Error-Funktion 28
4.1.2 Implementierung der Deadlock-Error-Funktion 29
4.1.3 Entwicklung der Temperatur-Error-Funktion 30
5 Implementierung des Watchdog-Prozesses 31
5.1 Handhabung der Timeout-Strategie 33
5.2 Umsetzung der Deadlock-Strategie 35
5.3 Implementierung der Temperatur-Strategie 35
6 Testausführung und Verhaltensbeobachtung des individuellen Watchdog-Mechanismus 35
6.1 Deadlock-Szenario 36
6.2 Timeout-Szenario 36
6.3 Temperature-Szenario 37
7 Kurzzeitige Programm-Ausführung und Beobachtung des Verhaltens der Watchdog-Mechanismen 39
8 Fazit zur Arbeit 43
Literaturverzeichnis 44

1 Einleitung

1.1Hintergrund und Bedeutung

In der Automobilindustrie spielt die Sicherheit eine entscheidende Rolle. Der Schutz von Menschen und Umwelt steht im Vordergrund und ist von größter Bedeutung. Ein wichtiger Aspekt der Sicherheit ist die Vermeidung von Systemfehlern in den eingebetteten Systemen, die in modernen Fahrzeugen verbaut sind. Diese Systeme umfassen die Steuerung des Motors, des Getriebes, des Brems- und Lenksystems sowie zahlreiche andere Funktionen. Aus diesem Grund müssen diese Systeme jederzeit zuverlässig funktionieren.

Ein Watchdog ist eine Überwachungseinheit, die in der Lage ist, Systemfehler zu erkennen und darauf zu reagieren. Sie ist so konzipiert, dass sie die Ausführung eines Programms oder Prozesses überwacht und im Falle eines Fehlers eingreift, um eine sichere Ausführung zu gewährleisten. In der Automobilindustrie spielt der Watchdog daher eine wichtige Rolle, um die Zuverlässigkeit und Sicherheit der eingebetteten Systeme zu gewährleisten.

Ein weiterer Hintergrund für die Bedeutung von Watchdogs in der Automobilindustrie ist die Tatsache, dass moderne Fahrzeuge immer mehr von eingebetteten Systemen abhängig sind. Diese Systeme sind notwendig, um die vielfältigen Funktionen und Leistungsmerkmale eines modernen Fahrzeugs zu unterstützen. Dabei müssen diese Systeme jedoch auch unter unterschiedlichsten Umweltbedingungen und in anspruchsvollen Fahrsituationen zuverlässig funktionieren. Ein Systemfehler in einem solchen kritischen Moment könnte verheerende Auswirkungen haben. Darüber hinaus haben moderne Fahrzeuge eine immer höhere Komplexität erreicht. Sie sind mit zahlreichen Sensoren, Aktuatoren und anderen Komponenten ausgestattet, die aufeinander abgestimmt und koordiniert werden müssen [6]. Der Watchdog spielt dabei eine wichtige Rolle, um sicherzustellen, dass alle Komponenten ordnungsgemäß funktionieren und aufeinander abgestimmt sind. Wenn ein Fehler in einem der Systeme auftritt, kann der Watchdog eingreifen und die korrekte Ausführung der anderen Systeme sicherstellen [6]. Ein weiterer wichtiger Aspekt ist die zunehmende Vernetzung von Fahrzeugen [16]. Moderne Fahrzeuge sind mit zahlreichen Kommunikationsschnittstellen ausgestattet, die eine nahtlose Integration in andere Systeme ermöglichen. Dadurch entstehen jedoch auch neue Risiken, da das Fahrzeug über diese Schnittstellen angreifbar wird. Der Watchdog kann dabei helfen, unautorisierte Zugriffe auf das System zu erkennen und abzuwehren.[16]

1.2 Zielsetzung der Arbeit

Das Ziel dieser Arbeit besteht darin, die verschiedenen Arten von Watchdogs zu untersuchen, die in der Automobilindustrie eingesetzt werden, und ihre Funktionsweise zu erklären. Ein weiteres Ziel ist es, einen praktischen Teil auf Linux zu erstellen, in dem ein Steuerprozess in C entwickelt wird. Zusätzlich wird ein Watchdog-Prozess entwickelt, der mit dem Steuerprozess kommuniziert und verschiedene Watchdog-Strategien bei auftretenden Fehlern evaluiert. Dazu gehören Endlosschleifen, Zeitüberschreitungen, Überhitzungen und Deadlocks.

2 Theoretischer Hintergrund

2.1 Grundlagen von Watchdog

2.1.1 Definition und Zweck von Watchdogs

Ein Watchdog dient als essenzieller Mechanismus, der in verschiedenen Bereichen eingesetzt wird, um die einwandfreie Funktionalität von Systemen oder Prozessen sicherzustellen. Der Begriff selbst leitet sich von der Tierhaltung ab, bei der ein Hund („dog“) über die Sicherheit der Herde wacht („watch“). In der technischen Anwendung wird ein Watchdog verwendet, um sicherzustellen, dass ein System oder Prozess ordnungsgemäß arbeitet und im Falle eines Fehlers oder einer Störung angemessene Maßnahmen ergreift. Der Watchdog ist eine Komponente [2], die entweder in Hardware oder Software innerhalb eines Computersystems integriert ist. Ihre Hauptaufgabe besteht darin, das System in einen vordefinierten Zustand zurückzusetzen, in der Regel durch einen Reset, wenn der Watchdog nicht regelmäßig zurückgesetzt wird.

Dieser Vorgang wird auch als „Nachtriggern“ bezeichnet. Der Watchdog überwacht kontinuierlich das System und gewährleistet dessen reibungslose Funktion. Sobald der Watchdog nicht mehr regelmäßig zurückgesetzt wird, signalisiert dies eine mögliche Störung oder einen Fehler im System, woraufhin er eingreift, um das System in einen stabilen Zustand zurückzubringen. Dadurch sichert der Watchdog die Zuverlässigkeit und Stabilität des Systems, indem er potenzielle Fehlerquellen erkennt und angemessen reagiert. Watchdogs stellen eine essenzielle Komponente einer widerstandsfähigen Softwarearchitektur dar. Im Folgenden werden einige potenzielle Szenarien aufgezeigt, die zu Fehlfunktionen in einem eingebetteten System führen können, wenn sie nicht gezielt zurückgesetzt werden. Ein mögliches Szenario beinhaltet einen fehlerhaft programmieren Interrupt-Handler. Wenn der Interrupt-Handler versäumt, die erforderlichen Schritte zur Beendigung der Interrupt-Quelle auszuführen, bleibt das Interrupt ausstehend und der Handler wird kontinuierlich aufgerufen. Dies führt zur Blockierung von niedrigem priorisiertem Code. Unter bestimmten Umständen kann ein Reset durch den Watchdog, sofern er nicht selbst niedrig priorisiert ist, den definierten Zustand wiederherstellen. Ein weiteres Szenario betrifft Endlosschleifen, die aufgrund unregelmäßiger Bedingungen entstehen. Wenn eine solche Endlosschleife in einem Interrupt-Handler auftritt, ähnelt sie dem zuvor beschriebenen Szenario. Eine dauerhafte Auslastung der CPU auf 100% durch eine Endlosschleife führt dazu, dass die CPU ausschließlich von dieser Schleife in Anspruch genommen wird, ohne anderen Prozessen die Möglichkeit zu geben, ausgeführt zu werden. In solchen Fällen werden in der

Regel Prozesse niedrigerer Priorität praktisch ausgebremst, während Prozesse höherer Priorität weiterhin laufen können. Je nachdem, wie die Aufgaben auf die verschiedenen Prozesse verteilt sind, kann diese Situation unter Umständen lange Zeit unentdeckt bleiben. [2]. Des Weiteren können Probleme durch fehlerhafte Prozesssynchronisation auftreten. Hierzu zählen Deadlocks, Lockouts, Convoy-Effekte und andere Probleme, die bei der nebenläufigen Programmierung auftreten können. Sofern der Prozess, der den Watchdog zurücksetzt, nicht betroffen ist, erkennt der Watchdog diese Bedingungen nicht. Daher ist es erforderlich, betroffene Prozesse durch Software-Watchdogs zu schützen. Durch den Einsatz von Watchdogs können diese potenziellen Probleme erkannt und das System in einen stabilen Zustand zurückgesetzt werden, um eine zuverlässige und kontinuierliche Funktionalität sicherzustellen [2].

2.1.2 Arten von Watchdogs

Es gibt eine Vielzahl von Watchdogs [2], die in verschiedenen Formen existieren und im Folgenden näher betrachtet werden:

1. Interne Hardware-Watchdogs: Die Mehrzahl der Prozessoren verfügt bereits über einen eingebauten Watchdog, der als eigenständiges Untermodul fungiert. Bei vielen ACPs wird dieses Modul beispielsweise als IWDG bezeichnet [2].

2. Externe Hardware-Watchdogs: Diese Watchdogs sind eigenständige integrierte Schaltkreise (ICs), die als eigenständige Einheiten operieren und die Regulierung durch das Hauptsteuergerät mithilfe von Kommunikationsschnittstellen wie I2C oder dem Systembus ermöglichen. Sie sind unmittelbar mit dem Prozessor-Neustartmechanismus verknüpft. Da es erforderlich ist, ein dediziertes Zeitsystem zu etablieren, werden diese Elemente oft in Kombination mit Taktgebebausteine eingesetzt. Sollte der Watchdog in Betrieb gehen und nicht rechtzeitig zurückgesetzt werden, führt dies zur Auslösung eines Prozessor-Neustarts über die Reset-Leitung [2].

Ein externer Watchdog kann entweder einen einzelnen Prozessor überwachen oder als Systemmonitor fungieren und mehrere Prozessoren und Subsysteme überwachen. In beiden Fällen ergeben sich mehrere Vorteile bei der Verwendung eines externen Watchdogs. Erstens ermöglicht er einen harten Neustart des Prozessors, um sicherzustellen, dass das System nach einem Fehlerzustand zuverlässig wiederhergestellt wird. Zweitens trägt ein externer Watchdog zu einem robusteren System bei, da er unabhängig von den internen Komponenten des Prozessors arbeitet und somit zusätzliche Sicherheitsschichten bietet. Drittens ist der Watchdog nicht auf den internen Oszillator des Prozessors angewiesen, was bedeutet, dass er auch dann funktioniert, wenn der Prozessor selbst nicht mehr ordnungsgemäß arbeitet. Schließlich ist der externe

Watchdog vom Prozess getrennt, den er überwacht. Dadurch kann er unabhängig von den internen Prozessen des überwachten Systems operieren und die Stabilität und Integrität des Gesamtsystems gewährleisten [5].

3. Software-Watchdogs: Dieser Mechanismus ermöglicht es, die Funktionalität eines Hardware-Watchdogs in Software umzusetzen.

Bei der Verwendung eines Software-Watchdogs in einem System sind verschiedene Aspekte von großer Bedeutung, die sorgfältig berücksichtigt werden sollten [5]. Insbesondere sollten die spezifischen Funktionen des Watchdogs bei der Arbeit mit internen Watchdogs eingehend betrachtet werden. Ein entscheidender Punkt ist, dass Watchdogs, die während der Laufzeit modifizierbar sind, ein gewisses Risiko bergen. Wenn der Code abstürzt, besteht die Möglichkeit, dass diese modifizierbaren Watchdogs deaktiviert werden und folglich den Prozessor nicht zurücksetzen können. Dies kann zu einer Beeinträchtigung der Systemsicherheit und Stabilität führen. Darüber hinaus ist es von essenzieller Bedeutung, dass ein Hardware-Reset durch den Watchdog ausgelöst wird, um sicherzustellen, dass der Prozessor ordnungsgemäß neu gestartet wird. Das bloße Zurückladen des Programmzählers reicht möglicherweise nicht aus, um den Prozessor vollständig zu reinitialisieren und die gewünschte Systemstabilität wiederherzustellen. Ein weiterer wichtiger Punkt ist, dass im Falle eines fehlenden Hardware-Resets sämtliche Peripheriegeräte manuell zurückgesetzt werden müssen. Dies kann zeitaufwendig und fehleranfällig sein, insbesondere in komplexen Systemen mit zahlreichen Peripheriekomponenten. Schließlich ist es wichtig zu beachten, dass eine einzelne I/O-Leitung für die Zuverlässigkeit des Watchdogs nicht ausreichend ist. Eine einfache Umschaltung oder ein Bitflip könnte den Timer zurücksetzen und somit zu unerwartetem Verhalten führen [2]. Es müssen geeignete Schutzmechanismen implementiert werden, um sicherzustellen, dass der Watchdog robust und zuverlässig arbeitet. Ein fortschrittlicherer Watchdog wird von NXP, LPC-Prozessorfamilie unterstützt [2]. Dieser sogenannte Window Watchdog Timer (WWDG) definiert sowohl ein Maximum als auch ein Minimum für die Nachtriggerzeit. Der Watchdog muss innerhalb eines festgelegten Zeitfensters nachgetriggert werden, sonst läuft er ab, auch wenn er zu früh nachgetriggert wird. Dies erkennt pathologische Zustände wie eine Endlosschleife des Watchdogs, die den Watchdog ständig nachtriggert. Das System funktioniert nicht mehr richtig, obwohl der Prozessor zufrieden ist [2]. Die Verwendung von Window Watchdogs erfordert mehr Sorgfalt, um Situationen wie Convoy-Effekte oder Lockouts zu vermeiden, bei denen der Watchdog verzögert wird und das Nachtriggern außerhalb des definierten Zeitfensters erfolgt. Window Watchdogs werden hauptsächlich in Echtzeitsystemen eingesetzt, wie z.B. Bremskontrollsys-

temen, die schnelle Reaktionen auf Ereignisse mit engen Toleranzgrenzen erfordern. Watchdogs, die Window Watchdogs unterstützen, können auch im Nicht-Fenster-Modus verwendet werden. Die höheren Anforderungen sind optional [2].

2.1.3 Funktionen des Watchdogs

Watchdogs müssen verschiedene Funktionen erfüllen, um ihre wichtige Überwachungsrolle auszuüben und potenzielle Bedrohungen zu erkennen und zu bekämpfen [2]. Ihre Aufgaben umfassen:

- **Einstellen der Nachtriggerzeit des Watchdogs:** Interne Watchdogs ermöglichen die Einstellung einer Nachtriggerzeit durch die Konfiguration eines Werts in einem entsprechenden Register. Dieser Wert kann ganzzahlige Bruchteile der Prozessortaktfrequenz umfassen [2].
- **Aktivieren des Watchdogs:** In der Regel erfolgt die Aktivierung des Watchdogs während der Initialisierungsphase des Prozessors. Nach der Aktivierung muss der Watchdog regelmäßig nachgetriggert werden. In den meisten Fällen kann ein bereits aktivierter Watchdog nicht mehr deaktiviert werden. Bei einigen Advanced Configuration and Power Interface (ACPI)-Implementierungen kann der Watchdog so konfiguriert werden, dass er automatisch nach einem Reset aktiviert ist, ohne eine explizite Aktivierung. Diese Konfiguration sollte jedoch nur verwendet werden, wenn gewährleistet ist, dass das Nachtriggern des Watchdogs zwischen einem Reset und dem vollständigen Betriebsbereitschaftsstatus der Software stets sicher gewährleistet ist. Es ist zu beachten, dass eine solche Funktion nur dann sinnvoll ist, wenn angenommen wird, dass ein erneuter Reset den Zustand behebt, der das fehlerhafte Ablaufen des Watchdogs verursacht hat. Andernfalls besteht die Gefahr einer Endlosschleife von wiederholten Resets [2].
- **Nachtriggerung des Watchdog:** Der Zeitpunkt der Nachtriggerung ist von entscheidender Bedeutung. Die Nachtriggerung muss gemäß der Definition in periodischen Abständen erfolgen, zu einem Zeitpunkt, an dem davon ausgegangen werden kann, dass das System reibungslos funktioniert. Es existieren zwei extreme Ansätze [2], die verfolgt werden können:

  1. **Das Nachtriggern im höchstpriorisierten Interrupt Handler**: Diese Lösung ist nicht optimal, da ein weniger hoch priorisierter Unterbrechungshandler in eine Endlosschleife geraten und das gesamte System zum Stillstand bringen kann, ohne dass der Watchdog dies erkennt.
  2. **Das Nachtriggern in einem sehr niedrig priorisierten Anwendungsprozess**: Diese Lösung birgt das Risiko, dass normale Bedingungen zu einer fehlerhaften Auslösung

des Watchdogs führen. Der optimale Zeitpunkt liegt irgendwo zwischen diesen Extremen und hängt vom Design des Systems und den spezifischen Anforderungen ab. Eine häufig gewählte Lösung besteht darin, einen Timer für das Betriebssystem zu verwenden. Das Verhalten dieses Timers hängt jedoch von der relativen Priorität der Aufgabe ab, da Betriebssystemtimer in der Regel im Kontext einer dedizierten

Timeraufgabe ausgeführt werden, die während des Systemstarts erzeugt wird. Es ist auch zu beachten, dass viele Systeme Komponenten enthalten, die die CPU für unvorhersehbare oder begrenzte Zeiträume beanspruchen, ohne dass dies als unregelmäßige Bedingung betrachtet wird. Ein Beispiel dafür sind Netzwerkschnittstellen. Wenn der Controller beispielsweise über Ethernet mit einem Netzwerk verbunden ist, können Broadcast-Stürme auftreten, die sowohl den Unterbrechungshandler des Ethernet-Controllers als auch die verarbeitenden Aufgaben stark beanspruchen und potenziell alle niedrigen priorisierten Operationen blockieren. Im Allgemeinen sollte der Zeitpunkt für das Nachtriggern eines Watchdogs so gewählt werden, dass möglichst viele tatsächliche Fehlerzustände erkannt werden, ohne die Gefahr einer fehlerhaften Auslösung bei zu langen Nachtriggerintervallen einzugehen. Die Auswahl des Intervalls für den Watchdog basiert auf verschiedenen Faktoren. Einerseits muss die längste reguläre Zeit berücksichtigt werden, die verstreichen kann, bevor der Watchdog erneut nachgetriggert werden muss. Dabei sollten auch Randbedingungen wie beispielsweise ein seltener, aber regulärer Firmwaredownload in Betracht gezogen werden. In solchen Fällen kann es sinnvoll sein, den Watchdog außerhalb des normalen periodischen Ablaufs manuell nachzutriggern. Andererseits muss eine akzeptable Zeitspanne definiert werden, in der das System in einem instabilen oder nicht funktionsfähigen Zustand verbleiben kann. Im schlimmsten Fall wird der Watchdog genau in der Mikrosekunde vor dem Auftreten des Fehlers nachgetriggert. Daher ergibt sich die maximale Gesamtzeit bis zur vollständigen Wiederherstellung des Systems aus einem Ablaufintervall und der Zeit, die das System nach einem Reset benötigt, um wieder funktionsfähig zu sein [2].

Des Weiteren spielt die Einschätzung eine Rolle, ob ein Watchdog-Reset als schwerwiegender Fehler für das System betrachtet wird. In einigen Architekturen wird lediglich eine ausreichende Stabilität für den Großteil der Laufzeit angestrebt, während Resets als akzeptabler Bestandteil betrachtet werden. In anderen Architekturen werden Geräteresets hingegen als relevante Ereignisse an den Host gemeldet und möglicherweise als kritischer Zustand betrachtet. Je nach Situation wird das Ablaufintervall so gewählt, dass gelegentliche „falsche Alarme“ im Sinne der Gesamtbetriebsfähigkeit akzeptiert

werden oder ein Reset erst dann riskiert wird, wenn ein Ablauf des Watchdog-Intervalls ein eindeutiges Zeichen für ein Fehlverhalten des Systems darstellt [2].

2.1.4 Konzept und Architektur eines Watchdogs

Grundsätzlich beruht ein Watchdog-Timer auf einem Zähler, der von einem bestimmten Startwert herunterzählt, bis er null erreicht. Die eingebettete Software wählt den Startwert des Zählers aus und startet ihn regelmäßig neu. Wenn der Zähler jemals null erreicht, bevor die Software ihn erneut startet, wird angenommen, dass die Software fehlerhaft ist. Infolgedessen wird das Zurücksetzungssignal des Prozessors aktiviert. Dadurch wird der Prozessor (und die darauf laufende eingebettete Software) neu gestartet, ähnlich wie wenn ein menschlicher Bediener die Stromversorgung unterbrochen und wieder eingeschaltet hätte [10].
Abbildung 1 veranschaulicht eine typische Konfiguration. Wie dargestellt, ist der Watchdog-Timer auf einem separaten Chip platziert, der extern zum Prozessor ist. Es ist jedoch auch möglich, den Watchdog-Timer innerhalb desselben Chips zu integrieren, auf dem sich die CPU befindet. Dies wird bei vielen Mikrocontrollern so umgesetzt. In beiden Fällen ist die Ausgabe des Watchdog-Timers direkt mit dem Zurücksetzungssignal des Prozessors verbunden. [10]

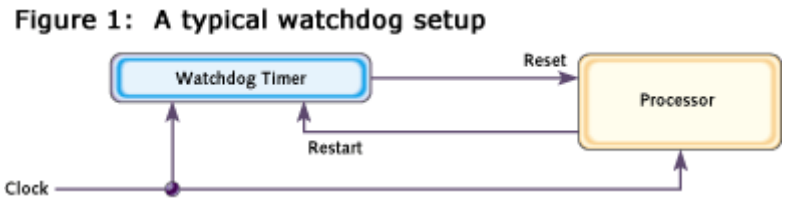

Abbildung 1: Eine typische Watchdog-Konfiguration [10]

Ein einfaches Beispiel wird im folgenden Code dargestellt. Hierbei handelt es sich um eine einzige Endlosschleife, die das gesamte Systemverhalten steuert. Diese Art von Softwarearchitektur ist typisch für viele eingebettete Systeme mit Low-End-Prozessoren und einem Verhalten, das auf einer einzigen Betriebsfrequenz basiert. Die Hardwareimplementierung des Watchdog-Timers erlaubt es, den Zählerwert durch ein Register festzulegen, das den Speicher abbildet.

```
uint16 volatile * pWatchdog =
        (uint16 volatile *) 0xFF0000;
main(void)
{
    hwinit();
    for (;;)
    {
        *pWatchdog = 10000;
        read_sensors();
        control_motor();
        display_status();
    }
}
```

Listing 1: Kicking the dog [10]

Unter der Annahme, dass die Schleife mindestens alle fünf Millisekunden einmal durchlaufen wird, um dem Motor regelmäßig neue Steuerungsparameter zuzuführen, und sofern der Zähler des Watchdog-Timers zu Beginn mit einem Wert von beispielsweise 10.000 initialisiert wird und keine Softwarefehler auftreten, wird der Watchdog-Timer niemals ablaufen. Die Software wird stets den Zähler neu starten, bevor er den Wert Null erreicht. [10].

Als nächstes werden drei Watchdog-Timer-Architekturen erörtert [9], die jeweils ansteigende Komplexität und Funktionalität aufweisen. Zunächst betrachten wir einen simplen, einstufigen Watchdog, der bedingungslos einen Neustart des Systems auslöst. Anschließend folgt ein zweistufiger Watchdog, der die Möglichkeit zur programmgesteuerten Wiederherstellung bietet. Abschließend wird ein dreistufiger Watchdog beschrieben, der eine programmgesteuerte Wiederherstellung ermöglicht und bei Fehlschlagen die Protokollierung von Zustands- und Debug-Informationen ermöglicht. Die grundlegendste Watchdog-Architektur umfasst einen einzigen Timer, der bei Ablauf direkt oder über eine Konditionierungsschaltung das System zurücksetzt, um einen Neustart zu erzwingen. Dabei ist das Timeout-Signal mit dem System-Reset-Eingang des Computers verbunden. Diese Architektur setzt voraus, dass der Systemneustart die Steuerausgänge in einen sicheren Zustand versetzt.

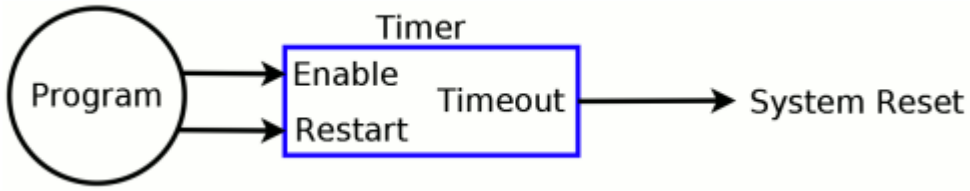

Abbildung 2: Diese Schaltung löst bei Zeitüberschreitung einen kontinuierlichen Reset aus [9]

In bestimmten Fällen werden bestimmte Computer heruntergefahren, wenn ein kontinuierliches Timeout-Signal auf den System-Reset-Eingang angewendet wird. Um diesem Problem entgegenzuwirken, kann ein Impulsgenerator verwendet werden, um einen kurzen Impuls zu erzeugen, der einen Systemneustart initiiert. Dieser Impulsgenerator erfüllt die Anforderung, einen spezifischen Impuls zur Auslösung des Systemneustarts bereitzustellen.

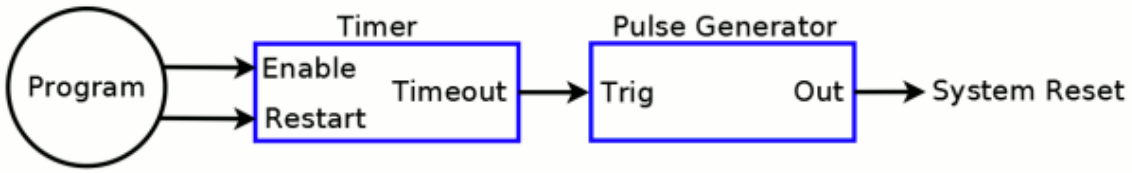

Abbildung 3: Diese Schaltung erzeugt Reset-Impulse bei Zeitüberschreitung [9]

Ausfallsichere Systeme mit mehrstufigen Watchdogs stellen sicher, dass bei einem Watchdog-Ereignis die Steuerausgänge des Computers prompt in sichere Zustände wechseln. Bei einem mehrstufigen Watchdog gibt es zwei oder mehr Timer, die zu unterschiedlichen Zeitpunkten Ereignisse signalisieren. Das erste Ereignis führt jedoch nicht sofort zu einem Neustart des Computers, sondern plant lediglich einen zukünftigen Neustart. Daher arbeitet ein mehrstufiger Watchdog mit einer speziellen Schaltung zusammen, die vor dem Neustart des Computers die Ausgänge in sichere Zustände schaltet. Um dies zu erreichen, kann ein separates Steuerungsreset-Signal verwendet werden, das bei Auftreten des ersten Watchdog-Ereignisses nur die Steuerschaltung (nicht den Computer selbst) zurücksetzt. Diese Methode ist einfach, aber es gibt einige Komplikationen und Mängel. Ein Steuerungsreset stellt die Ausgänge zwar auf ihre standardmäßigen Startzustände wieder her, aber diese Zustände sind möglicherweise nicht immer

ideal. Die optimalen sicheren Zustände können vom Gesamtsystemzustand abhängen. Zudem kann ein Steuerungsreset den Verlust wichtiger Zustandsinformationen verursachen, die für die Fehlerbehebung benötigt werden könnten, und auch den Betrieb von Schnittstellen beeinträchtigen, die normalerweise während eines Fehlerzustands weiterhin funktionieren könnten. Eine bessere Alternative besteht darin, zwei verschiedene Sätze von Steuerungszuständen, „Runmode“ und „Safemode“, zu definieren und einen Datenwähler zu verwenden, um die Zustandssätze unter der Kontrolle des Watchdogs auf die Ausgänge zu leiten. Das Programm kann die Runmode-Zustände jederzeit ändern, aber die Safemode-Zustände dürfen nur dann geändert werden, wenn sie durch einen speziellen Schreibschutzmechanismus genehmigt werden. In der Regel startet das Programm nach Festlegung der Safemode-Zustände, die einen vollständigen, maßgeschneiderten Satz sicherer Zustände für alle Ausgänge umfassen [9].

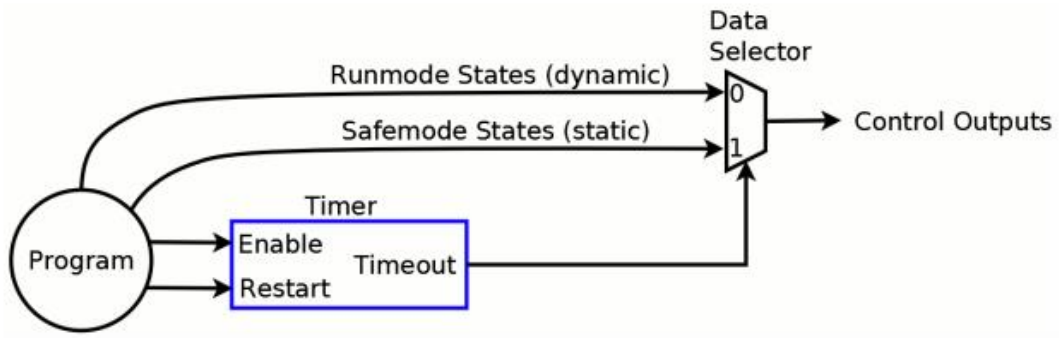

Abbildung 4: Ein flexibles, Watchdog-gesteuertes, ausfallsicheres System [9]

Um kostspielige plötzliche Systemneustarts oder den Verlust wichtiger Zustandsinformationen in Abhängigkeit von der Anwendungsart zu vermeiden, kann ein Watchdog mit mehreren Stufen eingesetzt werden. Der im Folgenden dargestellte Watchdog in zwei Stufen wechselt bei Ablauf von Timer 1 sofort die Steuerausgänge in sichere Zustände. Anstatt jedoch einen sofortigen Systemneustart auszulösen, plant er einen verzögerten Neustart und signalisiert dies dem Computer. Dadurch erhält das Programm Zeit, um sich von dem Fehler zu erholen, Zustands- oder Fehlerinformationen zu erfassen oder eine Kombination dieser Maßnahmen zu ergreifen. Bei einer erfolgreichen Wiederherstellung des Programms, wird der geplante Systemneustart abgesagt und ein kostspieliger Neustart des Systems vermieden [9].

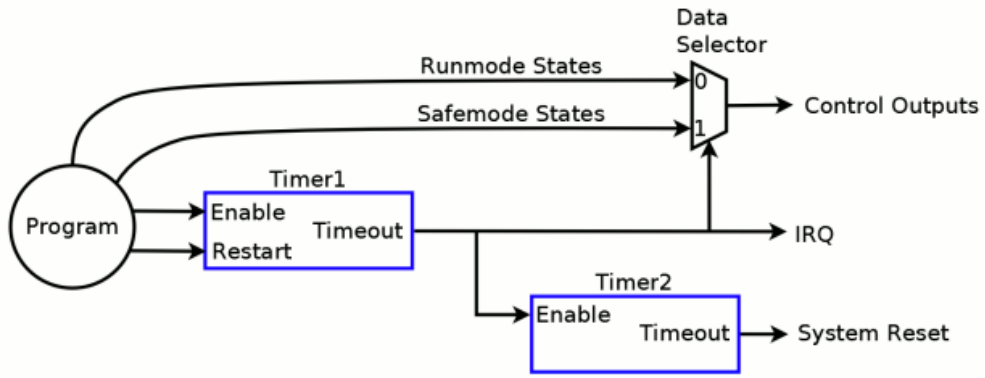

Abbildung 5: Zweistufiger Watchdog [9]

Während des normalen Betriebs gibt das Programm dem Timer 1 regelmäßig Impulse, um ein Ablaufzeitlimit zu verhindern. Solange Timer 1 nicht abgelaufen ist, ist Timer 2 deaktiviert und wird auf seinen Anfangswert gehalten, und die Steuerausgänge dürfen unter Programmsteuerung verändert werden. Bei einer Fehlermeldung läuft Timer 1 ab und zeitgleich werden die Ausgänge auf ausfallsichere Zustände umgeschaltet, woraufhin Timer 2 startet und über eine maskierbare Interruptanforderung (IRQ) einen Unterbrechungsdienst anfordert [9].

Wenn der Computer in der Lage ist, auf die IRQ zu reagieren, hat das Programm eine begrenzte Zeitspanne, um zu versuchen, sich von dem Fehlerzustand zu erholen oder, wenn der Fehler nicht korrigierbar ist, den Zustand und die Fehlerinformationen zu speichern. Bei erfolgreicher Wiederherstellung deaktiviert das Programm Timer1 und bricht somit den Systemneustart ab. Wenn der Computer nicht auf die IRQ reagieren kann oder eine Wiederherstellung unmöglich ist, wird ein Systemneustart ausgelöst, wenn Timer2 abläuft. Wenn das Programm die IRQ-Funktion niemals nutzt, kann das Ablaufzeitsignal von Timer1 an den NMI-Eingang, ein nicht maskierbarer Interrupt, anstatt an einen maskierbaren IRQ-Eingang geroutet werden. In diesem Fall benachrichtigt ein NMI den Computer, dass ein Systemneustart bevorsteht, und Timer 2 gibt ihm Zeit, Fehlerinformationen vor dem Neustart zu erfassen.

Der mehrstufige Watchdog, wie unten dargestellt, erweitert den Zweistufigen um einen dritten Timer und ermöglicht somit die Aufzeichnung von Debug-Informationen, falls eine erfolgreiche Wiederherstellung nicht möglich ist.

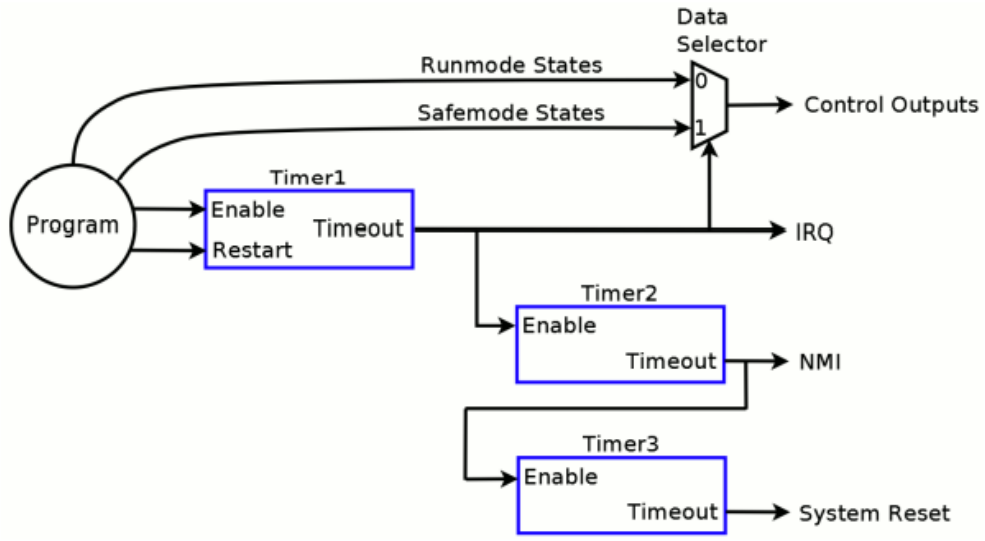

Abbildung 6: Dreistufiger Watchdog [9]

Während des normalen Betriebs gibt das Programm dem Timer1 regelmäßig Impulse, um eine Zeitüberschreitung zu verhindern. Während Timer 1 läuft, sind Timer 2 und Timer 3 deaktiviert und bleiben auf ihren Anfangswerten, und die Steuerausgänge dürfen unter Programmsteuerung verändert werden. Ähnlich wie im vorherigen Beispiel schaltet eine Zeitüberschreitung von Timer 1 die Ausgänge in sichere Zustände um, startet Timer2 und fordert einen Interrupt-Service an. Wenn der Computer in der Lage ist, auf die IRQ zu reagieren, wird das Programm versuchen, sich von dem Fehlerzustand zu erholen. Bei erfolgreicher Wiederherstellung deaktiviert das Programm Timer 1 und bricht weitere Korrekturmaßnahmen ab. Wenn der Computer nicht auf die IRQ reagieren kann, zu lange braucht, um zu reagieren, oder eine erfolgreiche Wiederherstellung unmöglich ist, wird Timer 2 ablaufen. Dadurch wird Timer 3 gestartet und ein nicht maskierbarer Interrupt-Request ausgelöst, um anzuzeigen, dass ein Systemneustart bevorsteht. Wenn der Computer entsprechend konfiguriert ist und auf den NMI reagieren kann, wird er wichtige Fehlerinformationen (z. B. Crash-Dump) protokollieren. Das System wird neu gestartet, wenn Timer 3 abläuft [9].

2.2 ISO 26262 und AUTOSAR

Der ISO-26262-Standard wurde 2011 von der ISO-Organisation entwickelt und hat zum Ziel, grundlegende Prinzipien der funktionalen Sicherheit in der Automobilentwicklung zu etablieren. Ein wichtiger Bestandteil dieses Standards sind die Sicherheitsintegritätslevel (ASIL), die in den Stufen A, B, C und D definiert sind. Insbesondere die Stufen C und D beschäftigen sich mit den kritischsten Szenarien [1]. Die ISO 26262 stellt ein Konzept zur Gewährleistung der

funktionalen Sicherheit bereit, wobei das Ablaufüberwachungssystem, auch bekannt als Program Flow Monitoring durch den Watchdog Manager, ein essenzieller Bestandteil dieses Konzepts ist [18]. Dieser Mechanismus spielt eine bedeutende Rolle bei der Überwachung des Programmablaufs und trägt zur Sicherheit des Systems bei. Gleichzeitig definiert AUTOSAR eine Softwareplattform und einen Middleware-Standard für eingebettete Systeme in der Automobilindustrie. AUTOSAR wird in Zusammenarbeit mit einem breiten Konsortium von Automobilherstellern, Zulieferern und Werkzeuganbietern entwickelt. Die Mitglieder dieses Konsortiums arbeiten zusammen, um die AUTOSAR-Plattform zu entwickeln und zu standardisieren, während sie gleichzeitig konkurrierende Implementierungen von Softwarestapeln und Entwicklungstools anbieten. Das Hauptziel von AUTOSAR besteht darin, die Entwicklung von hardwareunabhängigen Anwendungsfunktionen zu erleichtern, die in jeder AUTOSAR-Umgebung ausgeführt werden können. Dies wird durch die Verwendung von AUTOSAR-Schnittstellen ermöglicht, die die Kommunikation zwischen verschiedenen Softwaremodulen ermöglichen. Die AUTOSAR-Runtime-Umgebung stellt eine einheitliche API zur Kommunikation zwischen den Komponenten der Anwendungssoftware bereit, unabhängig davon, ob sie auf demselben Steuergerät oder auf verschiedenen Steuergeräten ausgeführt werden. Ein weiteres wichtiges Merkmal von AUTOSAR ist die Definition von Anforderungen für Basissoftwarekomponenten, um eine Abstraktion von Steuergeräten und Mikrocontrollern zu ermöglichen. Durch diese Abstraktion können verschiedene Softwaremodule auf unterschiedlichen Steuergeräten bereitgestellt werden, ohne dass eine Änderung des Anwendungscodes erforderlich ist [11].

Abbildung 11 und Abbildung 12 [3] veranschaulichen den Mechanismus zur Überwachung des Programmflusses in einer Multicore-Architektur. Dabei stehen verschiedene Herausforderungen im Zusammenhang mit den Kernadressen im Fokus, die folgendermaßen gelöst werden müssen: Um eine zuverlässige Überwachung zu gewährleisten, ist es entscheidend, sicherzustellen, dass in den Sicherheitseinrichtungen der Kerne keine gegenseitige Überprüfung stattfindet. Dadurch wird verhindert, dass die Kernsysteme sich bei der Überwachung des Programmflusses gegenseitig beeinflussen. Ein weiterer wichtiger Aspekt besteht darin, die Verfügbarkeitsanfragen für Überprüfungspunkte in einem Kern zu platzieren und den Programmflussüberwachungsmechanismus eines anderen Kerns zur Durchführung dieser Anfragen aufzurufen. Durch diese koordinierte Kommunikation zwischen den Kernen wird sichergestellt, dass alle relevanten Informationen erfasst werden und ein konsistenter Überblick über den Programmfluss gewährleistet ist. Darüber hinaus ist eine sorgfältige Synchronisation der Kernverbindungen erforderlich, um einen akzeptablen Jitter zwischen den Kernen sicherzustellen. Dies

bezieht sich auf die minimale zeitliche Abweichung zwischen den Kernen, um einen reibungslosen und synchronisierten Ablauf zu gewährleisten. Schließlich muss ein Watchdog-Reaktionsmechanismus aktiviert werden, wenn ein schwerwiegender Fehler im Programmflussüberwachungsmechanismus eines Kerns auftritt. Dabei übernimmt der Mechanismus zur Programmflussüberwachung eines anderen Kerns die Aktivierung dieses Reaktionsmechanismus. Dieser Prozess ermöglicht eine schnelle Reaktion auf Fehler und unterstützt die Stabilität des Systems. Die oben genannten Maßnahmen stellen sicher, dass der Programmfluss in einer Multicore-Architektur effektiv überwacht wird. Durch die Vermeidung gegenseitiger Überprüfung in den Sicherheitseinrichtungen, die koordinierte Platzierung von Verfügbarkeitsanfragen, die Synchronisation der Kernverbindungen und die Aktivierung des Watchdog-Reaktionsmechanismus können potenzielle Fehler erkannt und entsprechend reagiert werden.

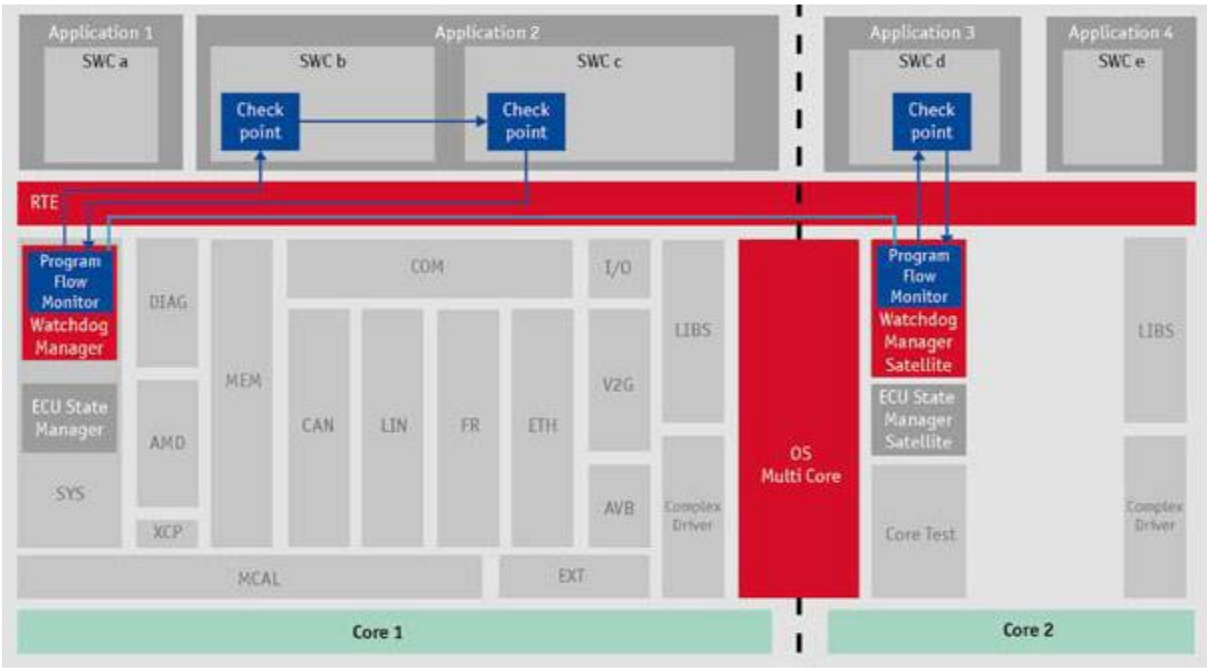

Abbildung 7: Überwachungsmechanismus mit Watchdog-Flow-Monitoring in der geschichteten AUTO-SAR-Architektur [3]

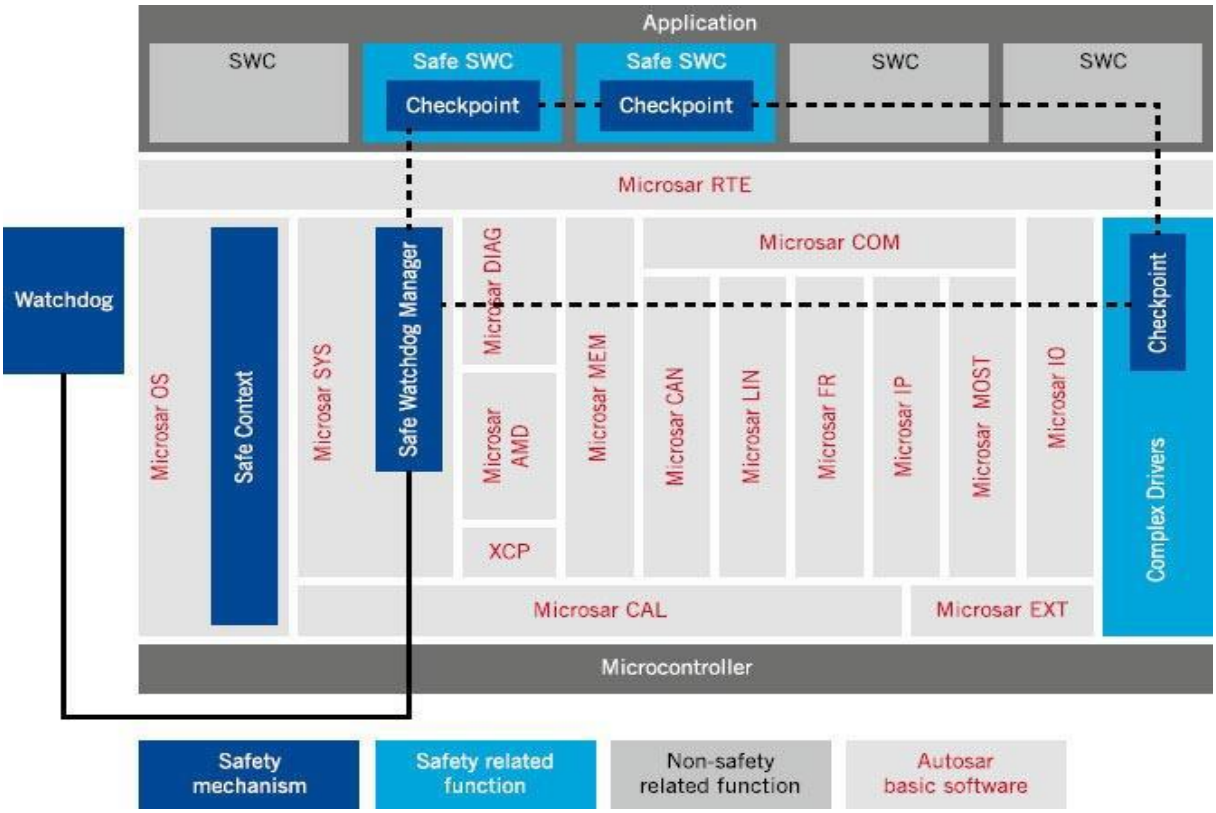

Abbildung 8:Überwachungsmechanismus mit Watchdog-Flow-Monitoring in der geschichteten Microsar-Architecture [3]

2.2.1 Features zur Health-Monitoring für Watchdogs

Gemäß den Anforderungen des AUTOSAR wird das Health Monitoring zur Unterstützung des Timeout-Verhaltens von Watchdogs eingesetzt. Diese Watchdogs dienen dazu, sicherzustellen, dass bestimmte Werte innerhalb eines definierten Zeitlimits geschrieben werden.

Zusätzlich zum Timeout-Verhalten sollen im Rahmen des Health Monitoring auch Window-Watchdogs unterstützt werden. Bei diesen Watchdogs müssen die Werte innerhalb eines definierten Zeitfensters korrekt geschrieben werden. Es werden verschiedene Modi der Hardware-Watchdogs unterstützt, darunter Normal, Niedrig, Aus oder Sleep-Modus. Das Health Monitoring soll in der Lage sein, unterschiedliche Realisierungen der Watchdogs zu unterstützen. Dies umfasst interne Hardware-Watchdogs im Mikrocontroller, externe Hardware-Watchdogs, separate dedizierte Chips (Application-Specific Integrated Circuits, ASICs) oder Anwendungen auf separaten Mikrocontrollern. Des Weiteren können mehrere Watchdogs unterstützt werden, die entweder vom gleichen oder unterschiedlichen Typ sind und die gleiche oder unterschiedliche Konfiguration aufweisen können. Das Health Monitoring sollte die Fähigkeit besitzen, Überwachungsfehler zu erkennen und entsprechende Fehlerreaktionen zu ermöglichen [15].

Darüber hinaus soll das Health Monitoring auch Frage-Antwort-Watchdogs unterstützen. Hierbei hängt die Antwort des Watchdogs von der gestellten Frage und den aktuellen Ergebnissen des Health Monitoring ab [15]. Der DRV3205-Q1, ein von Texas Instruments entwickelter Gate-Treiber, ist ein hochspezialisiertes elektronisches Gerät, das im Automobilbereich Verwendung findet. Dieses Gate-Gerät spielt eine entscheidende Rolle in der Steuerung von Motoren in Fahrzeugen und gewährleistet die sichere und effiziente Leistungsübertragung. Dieser hochentwickelte Treiber implementiert das Frage-Antwort-Watchdog-Feature, das zum Zweck der Überwachung einer externen Mikrocontroller-Einheit (MCU) dient. Die Funktionsweise des Q&A Watchdog-Features ist periodisch und basiert auf der Übertragung spezifischer Nachrichtensequenzen über das Serial Peripheral Interface. Nach einer Anfrage von der MCU, liefert der DRV3205-Q1 über das SPI eine Identifikation (Token oder Frage), die im WDT_TOKEN_VALUE-Register gespeichert wird. Die MCU führt daraufhin eine vordefinierte Abfolge von Berechnungen mit dem erhaltenen Identifikationswert aus und sendet die errechneten Antworten über SPI zurück, wobei sie diese im WDT_ANSWER-Register ablegt. Der DRV3205-Q1 überwacht dabei, ob die MCU die Antworten innerhalb der vorgegebenen Zeitspannen zurückschickt und ob diese korrekt sind. Während des Watchdog-Prozesses können verschiedene Arten von Antwortereignissen auftreten. Ein gutes Ereignis tritt auf, wenn die Antworten in der richtigen Reihenfolge und innerhalb des korrekten Zeitfensters gesendet werden. Dadurch verringert sich der Fehlerzähler des Watchdogs WD_FAIL_CNT, und ein neues Token wird generiert, um den Prozess von Neuem zu starten. Im Gegensatz dazu führt ein schlechtes Ereignis dazu, dass die MCU die Watchdog-Sequenz fehlerhaft ausführt. Dies erhöht den Watchdog-Fehlerzähler WD_FAIL_CNT, und das Token für den nächsten Frame bleibt unverändert. Überschreitet der WD_FAIL_CNT-Zähler den Wert von WD_FAIL_MAX, wird die konfigurierte Fehlerbedingung ausgeführt. Ein weiteres Szenario ist der falsche Antwortwert. Dabei führt ein falscher Wert im WDT_ANSWER-Register für eine der Antworten zu einem Token-Fehler und einem Sequenzfehler. Der Token-Fehler bleibt aktiv, bis die nächste Antwort gegeben wird, und der Sequenzfehler bleibt gesetzt, bis die nächste korrekte Sequenz ausgeführt wird. Ähnlich verhält es sich bei der falschen Antwortsequenz, bei der eine empfangene Token-Antwort in der falschen Reihenfolge ist. Die Reaktion ähnelt dem Fall des falschen Antwortwerts, außer dass die Token-Fehler-Flagge (TOKEN_ERR) für jede Antwort gesetzt bleibt, die sich in der Reihenfolge befindet. Des Weiteren kann es zu Zeitfensterfehlern kommen, wenn Antworten außerhalb des richtigen Zeitfensters gesendet werden, Beispiele hierfür sind das Senden einer Antwort vor dem erwarteten Zeitpunkt während des geschlossenen Zeitfensters,

das Empfangen einer vierten Antwort im offenen Zeitfenster, obwohl nur drei Antworten erwartet wurden, oder das Empfangen einer Antwort nach Abschluss der Transaktion. Wenn die vierte Antwort im offenen Zeitfenster empfangen wird, generiert das DRV3205-Q1-Gate-Treiber eine Token-Früh-Flagge TOKEN_EARLY, die nicht gelöscht wird, bis das nächste gute Antwortereignis eintritt [14]. Ein Ereignis ohne Antwort tritt auf, wenn die MCU die mit dem Watchdog verbundene SPI-Kommunikation für die Dauer des Watchdog-Zeitfensters aussetzt. Während dieses Ereignisses wird die TIME_OUT Flagge gesetzt, die von der MCU-Software verwendet werden kann, um die Watchdog-Ereignisse wieder auf die erforderliche Watchdog-Zeitabstimmung zu synchronisieren. Die Token-Generierung erfolgt durch einen Watchdog-Timer mithilfe einer Linear-Feedback-Shift-Register (LFSR)-Schaltung, wodurch zyklische Token-Werte erzeugt werden. Die LFSR-Gleichung ist standardmäßig eingestellt und erzeugt insgesamt 15 Werte für jeden Zyklus. Der Anfangswert (Seed) des LFSR ist auf einen Standardwert vorbeladen und wird nach jedem LFSR-Zyklus auf einen neuen Wert gesetzt. Nach jedem guten Ereignis erhöht sich ein interner Vier-Bit-WDT-Tokenzähler. Die Kombination dieses Zählerwerts und des LFSR-Ausgangs wird verwendet, um den Token-Wert zu erstellen. Wenn der WDT-TOKEN-Zähler überläuft, wird der Seed-Wert basierend auf der LFSR-Gleichung verschoben [14].

2.2.2 Watchdog-Stack in AUTOSAR

Die vorliegende Architektur [12] in Abbildung 10 besteht aus drei primären Modulen: dem Watchdog Manager (WdgM) in der Service-Schicht, der Watchdog Interface (WdgIf) in der ECU-Abstraktionsschicht und dem Watchdog Driver (WDG) in der MCAL-Schicht.

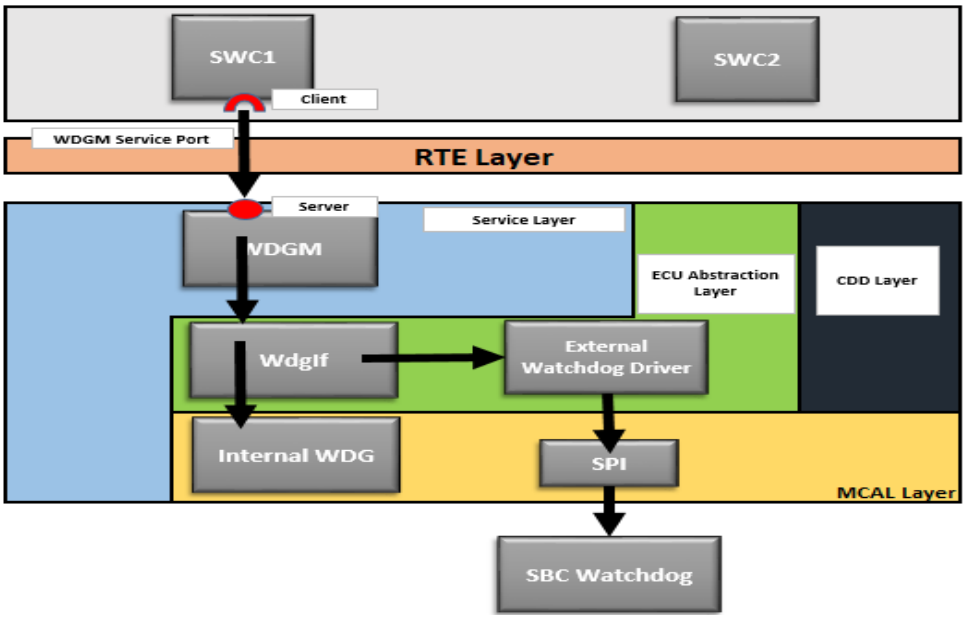

Abbildung 9: AUTOSAR Watchdog Stack [12]

.

2.2.2.1 Watchdog Manager

Der Watchdog Manager fungiert als zentrales Modul innerhalb der Watchdog-Stack-Architektur von AUTOSAR und überwacht die Ausführung des Programms unabhängig von der Auslösung von Hardware-Watchdog-Entitäten. Er überwacht die Ausführung einer konfigurierbaren Anzahl von sogenannten überwachten Entitäten. Wenn er eine Verletzung der zeitlichen und/oder logischen Einschränkungen bei der Programmausführung erkennt, ergreift er eine Reihe von konfigurierbaren Maßnahmen, um sich von diesem Fehler zu erholen.
Der Watchdog Manager bietet drei Mechanismen [12] zur Unterstützung des Health Monitorings:
1. Alive Supervision – zur Überwachung des Timings periodischer Software.
Periodische überwachte Entitäten unterliegen Beschränkungen hinsichtlich der Anzahl ihrer Ausführungen innerhalb eines bestimmten Zeitraums. Durch die Alive Supervision überprüft der Watchdog Manager regelmäßig, ob die Meilensteine einer überwachten Entität innerhalb der festgelegten Grenzen erreicht wurden. Dies bedeutet, dass der Watchdog Manager überprüft, ob eine überwachte Entität weder zu häufig noch zu selten ausgeführt wird [12].
2. Deadline Supervision – zur Überwachung aperiodischer Software.
Aperiodische oder episodische überwachte Entitäten unterliegen individuellen Beschränkungen hinsichtlich des Zeitablaufs zwischen zwei Meilensteinen. Durch die Deadline Supervision überprüft der Watchdog Manager den zeitlichen Ablauf der Übergänge zwischen zwei Meilensteinen einer überwachten Entität. Das bedeutet, dass der Watchdog Manager prüft, ob bestimmte Schritte in einer überwachten Entität eine Zeit benötigen, die innerhalb der konfigurierten Mindest- und Höchstwerte liegt [12].
3. Logical Supervision – zur Überwachung der Korrektheit der Ausführungsreihenfolge [12].
Die logische Supervision, die gemäß ISO 26262 in Teil 6 empfohlen ist [11], konzentriert sich auf Kontrollflussfehler, die von der gültigen (d. h. programmierten/kompilierten) Programmsequenz abweichen, während die Anwendung fehlerfrei ausgeführt wird. Ein fehlerhafter Kontrollfluss tritt auf, wenn eine oder mehrere Programmieranweisungen entweder in der falschen Reihenfolge verarbeitet werden oder überhaupt nicht verarbeitet werden. Kontrollflussfehler können zu Datenbeschädigung, Resets des Mikrocontrollers oder Verletzungen der Fehlerstummschaltung führen. Im Kontrollflussgraphen erfordert dies eine Überprüfung jedes Mal, wenn die überwachte Entität einen neuen Meilenstein meldet, ob zwischen dem vorherigen Meilenstein und dem gemeldeten ein konfigurierter Übergang besteht.

2.2.2.2 Watchdog Interface

Die Watchdog Interface befindet sich in der ECU-Abstraktionsschicht. Die Aufgabe der Watchdog Interface besteht darin, Befehle/Daten/Aufrufe vom Watchdog Manager an den darunterliegenden Watchdog-Treiber weiterzuleiten. Im System können mehrere Watchdog-Treiber vorhanden sein (intern oder extern). Die Aufgabe des WdgIf besteht darin, die Anfragen vom WdgM an die entsprechende WDG-Treibereinheit weiterzuleiten [12].

2.2.2.3 Watchdog Driver

Der Watchdog-Treiber dient der Initialisierung des Hardware-Watchdogs, der Auslösung des Watchdogs und der Änderung des Watchdog-Modus (Slow/Fast/OFF). Im Slow-Modus ist die Timeout-Zeit des Watchdogs hoch, beispielsweise 1000 ms, und wird typischerweise während des Systemstarts angewendet. Im Fast-Modus hingegen ist die Timeout-Zeit niedrig, beispielsweise 100 ms, und wird üblicherweise zur Laufzeit verwendet. Im OFF-Modus ist die Watchdog-Funktionalität deaktiviert, vorausgesetzt, dass die Konfiguration WdgDisableAllowed auf „true“ gesetzt ist. Wenn der interne Watchdog nicht verwendet wird und stattdessen ein externer Watchdog eingesetzt wird, befindet sich der Treiber für den externen Watchdog in der Abstraktionsschicht an Bord (ECU-Abstraktion). Dieser Treiber erfordert weitere MCAL-Treiber (zum Beispiel SPI), um mit dem externen Watchdog zu kommunizieren. Der AUTOSAR-Watchdog-Treiber umfasst einen Zähler, der beim Systemstart und bei der Initialisierung des WDG (mittels der Wdg_Init () -Funktion) auf null gesetzt wird. Die Initialisierung erfolgt durch den AUTOSAR-Watchdog-Treiber, um den Hardware-Watchdog-Treiber zu initialisieren. Der Watchdog-Zähler beginnt daraufhin zu zählen. Sobald der Zähler einen bestimmten Wert erreicht, der der Watchdog-Timeout-Periode entspricht, wird ein Reset durch den Watchdog ausgelöst. Um unerwünschte Resets zu vermeiden, muss die Software den Watchdog vor Erreichen dieses spezifischen Werts auslösen [13]. Bei der Konfiguration des Watchdog-Treibers kann festgelegen werden, welcher Initialisierungsmodus (Slow/Fast) verwendet werden soll. Später ändert der Watchdog-Manager zur Laufzeit den Modus (Slow/Fast) entsprechend der Konfiguration. Wdg_SetMode () ist die Funktion, die vom Watchdog-Treiber bereitgestellt wird. Um den Watchdog zur richtigen Zeit auszulösen, wird in AUTOSAR empfohlen, den GPT-Timerdienst zu verwenden. Wenn man beispielsweise den Watchdog nach 60 ms auslösen möchte, kann man den GPT-Timer auf 60 ms konfigurieren und die GPT-Interrupt Service Routine (ISR) verwenden, um den Watchdog zu einem bestimmten Zeitpunkt auszulösen. Wie bereits erwähnt, den Watchdog kann mithilfe der GPT-Dienste (ISR) ausgelöst werden. Bei der Auslösung des Watchdogs muss jedoch die Auslösebedingung berücksichtigt werden. Die Auslösebedingung wird vom Watchdog-Manager aktualisiert. Wenn die Auslösebedingung für den

HW-Watchdog erfüllt ist, indem der Trigger-Zähler einen Wert größer als Null aufweist, erfolgt eine Auslösung des Watchdogs. Andernfalls erfolgt keine Auslösung, was zu einem Überlauf des Watchdog-Zählers und einem anschließenden Watchdog-Reset führt. Das heißt, der Watchdog-Manager legt die Auslösebedingung fest, und basierend darauf entscheidet der Watchdog-Treiber, ob der Watchdog ausgelöst werden soll oder nicht. Der Watchdog-Treiber stellt die Funktion Wdg_SetTriggerCondition(timeout) bereit, die es dem Watchdog-Manager ermöglicht, die Auslösebedingung bzw. den Trigger-Zähler festzulegen. Dabei kann der Trigger-Zähler einen Wert von 0 oder größer aufweisen. In Abbildung 11 wird die Sequenz-Diagramm für das Triggern des Watchdogs gezeigt. Der Watchdog-Manager legt die Auslösebedingung basierend auf der Überwachung fest, die vom Watchdog-Manager durchgeführt wird. Sobald der Interrupt auftritt (d.h. der GPT-Interrupt), löst der WDG-Treiber basierend auf der Auslösebedingung entweder den HW-Watchdog aus oder löst den HW-Watchdog nicht aus. Das Triggern des externen Watchdogs erfolgt über den SPI/DIO-Treiber [13].

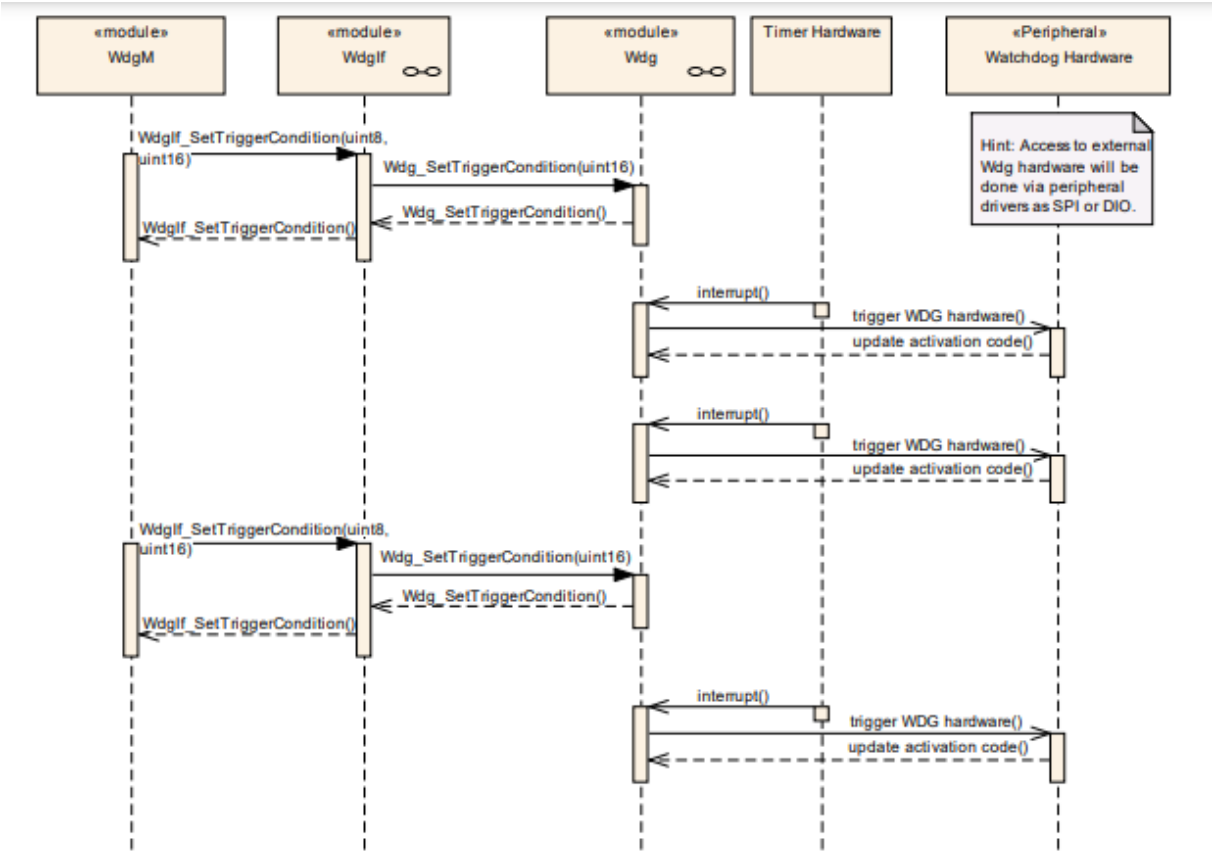

Abbildung 10: Triggern des Watchdogs [13]

3 Methodik und Implementierung

3.1 Entwicklungsumgebung

Die praktische Umsetzung konzentriert sich nicht auf die konkrete Implementierung auf einem Mikrocontroller, da dies eine erhöhte Sicherheit und ein Echtzeitbetriebssystem wie RTOS [17] erfordern würde. Stattdessen wird der praktische Aspekt auf Linux realisiert, um die Konzepte und Ideen hinter der Prozesskommunikation zu veranschaulichen. Das vorrangige Ziel besteht darin, zu demonstrieren, wie Watchdogs auf einem Controller reagieren können, um potenzielle Probleme zu bewältigen.

3.2 Auswahl der Kommunikationsmethode

Im Rahmen der praktischen Umsetzung wurde die Kommunikationsmethode zwischen dem Watchdog-Prozess und dem Controller-Prozess sorgfältig ausgewählt. Nach einer gründlichen Analyse und Bewertung verschiedener Optionen fiel die Entscheidung auf die Verwendung einer Message Queue als geeignete Kommunikationsmethode. Die Nutzung einer Message Queue bietet eine effiziente und zuverlässige Möglichkeit für die Kommunikation zwischen den beiden Prozessen [8]. Durch die Message Queue können Nachrichten in einer Warteschlange gespeichert werden, was den Watchdog-Prozess und Controller-Prozess unabhängig voneinander arbeiten lässt. Der Controller-Prozess kann relevante Informationen und Statusaktualisierungen in die Message Queue senden, während der Watchdog-Prozess diese Nachrichten aus der Warteschlange empfängt und entsprechend reagiert. Die Verwendung einer Message Queue bietet mehrere Vorteile für die Kommunikation zwischen den beiden Prozessen. Erstens ermöglicht sie eine asynchrone Kommunikation, bei der die Prozesse nicht direkt miteinander interagieren müssen. Dadurch wird die Flexibilität und Effizienz bei der Aufgabenausführung erhöht. Zweitens bietet die Message Queue eine Pufferungsfunktion, sodass Nachrichten, die während der Abwesenheit des Controller-Prozesses empfangen werden, nicht verloren gehen. Der Controller-Prozess kann die Nachrichten bei Bedarf aus der Warteschlange abrufen, um sicherzustellen, dass keine wichtigen Informationen verloren gehen. Darüber hinaus ermöglicht die Verwendung einer Message Queue auch eine gewisse Entkopplung zwischen den Prozessen. Jeder Prozess kann seine eigene Geschwindigkeit und Priorität beibehalten, ohne direkt von den Aktionen des anderen Prozesses abhängig zu sein. Dies führt zu einer verbesserten Skalierbarkeit und Flexibilität bei der Entwicklung und Wartung des Systems [8].

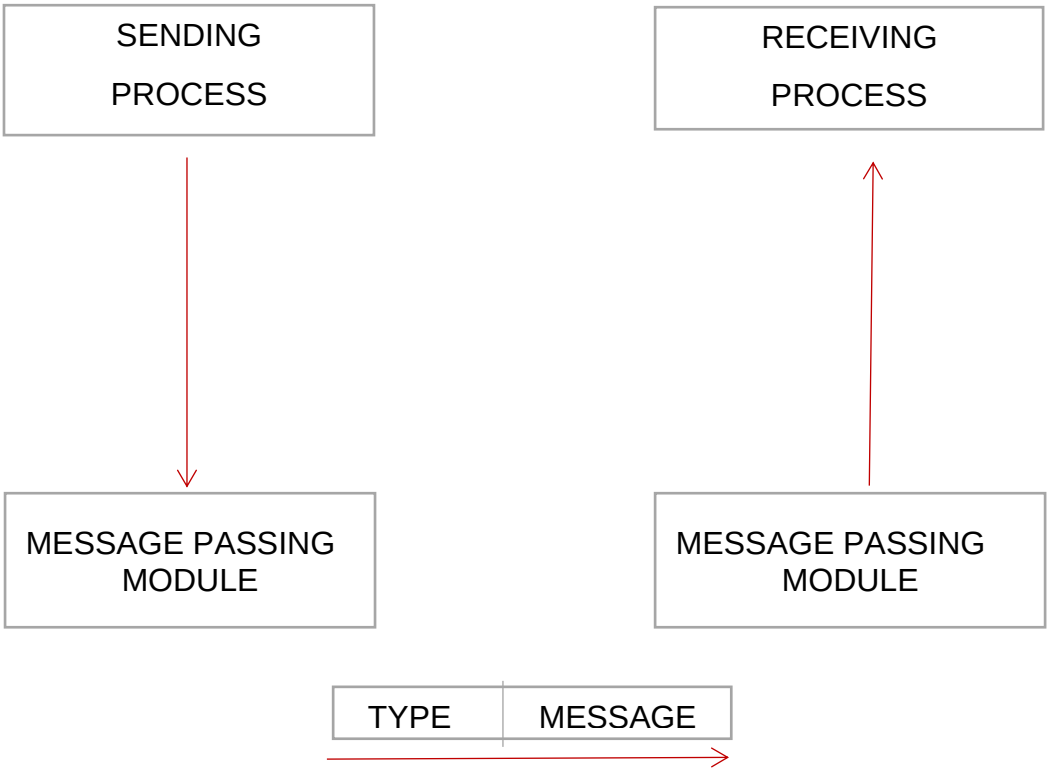

Abbildung 11: Message Queue [8]

Eine Nachrichtenwarteschlange besteht aus einer verketteten Liste von Nachrichten, die im Kernel des Betriebssystems gespeichert werden. Jede Nachricht wird durch eine eindeutige ID identifiziert. Für die Erstellung einer neuen Warteschlange oder den Zugriff auf eine bestehende Warteschlange wird die Funktion msgget() verwendet. Diese Funktion liefert die Warteschlangen-ID zurück, die für weitere Operationen genutzt werden kann.

```
// Create a message queue
messageQueueID = msgget(key, IPC_CREAT | 0666)
```

Listing 2: Eine Nachricht erstellen

Der Parameter key fungiert als eindeutiger Identifikator für die Nachrichtenwarteschlange. Das Flag IPC_CREAT gibt an, dass die Warteschlange erstellt werden soll, falls sie nicht vorhanden ist. Das Flag 0666 definiert die Zugriffsrechte für die Warteschlange, erlaubt Lesen und Schreiben für alle Benutzer.

Um eine Nachrichtenwarteschlange zu entfernen und aus dem System zu löschen, wird die Funktion msgctl() verwendet.

```
// Remove the message queue
msgctl(messageQueueID, IPC_RMID, NULL);
```

Listing 3: Eine Nachrichtenwarteschlange entfernen

Der erste Parameter messageQueueID ist die eindeutige ID der Nachrichtenwarteschlange, die gelöscht werden soll. Diese ID wurde zuvor beim Erstellen oder Öffnen der Warteschlange erhalten. Das Flag IPC_RMID gibt an, dass die Warteschlange entfernt werden soll. Es signalisiert dem System, dass die angegebene Nachrichtenwarteschlange gelöscht werden soll. Der dritte Parameter NULL gibt an, dass keine weiteren spezifischen Informationen für die Löschung der Warteschlange benötigt werden. Es ist möglich, zusätzliche Optionen oder Daten zu übergeben, aber in diesem Fall sind keine weiteren Informationen erforderlich. Zur Übermittlung einer Nachricht in die Message Queue wird die Funktion msgsnd() verwendet. Diese Funktion erfordert die Angabe des Nachrichtentyps, der Nachrichtenlänge und der eigentlichen Nachrichtendaten. Die übermittelte Nachricht wird am Ende der Warteschlange hinzugefügt.

```
// Send Message
msgsnd(messageQueueID, &message, sizeof(Error), 0);
```

Listing 4: Eine Nachricht senden

Das Abrufen von Nachrichten aus der Warteschlange erfolgt mit der Funktion msgrcv(). Hierbei haben wir die Flexibilität, Nachrichten basierend auf ihrem Typ abzurufen. Das heißt, man kann selektiv bestimmte Nachrichtentypen auswählen und verarbeiten, anstatt die Nachrichten in der Reihenfolge zu erhalten, in der sie hinzugefügt wurden.

```
// Receive message
msgrcv(messageQueueID, &message, sizeof(Error), 1, 0);
```

Listing 5: Eine Nachricht empfangen

3.3 Erzeugung der zwei Prozesse

Die Funktion fork() wird in Linux- und Unix-Betriebssystemen verwendet, um einen neuen Prozess zu erzeugen, der als Kindprozess bezeichnet wird und parallel zum aufrufenden Prozess, dem Elternprozess, ausgeführt wird [4]. Nach dem Aufruf von fork() führen sowohl der Eltern- als auch der Kindprozess die nächste Anweisung nach dem Aufruf aus.

Der Kindprozess teilt denselben Programmzähler, dieselben CPU-Register und dieselben geöffneten Dateien mit dem Elternprozess. Es sind keine Parameter erforderlich, und die Funktion gibt einen ganzzahligen Wert zurück. Durch die gemeinsame Nutzung dieser Ressourcen können Eltern- und Kindprozesse parallel arbeiten und auf denselben Datenbestand zugreifen.

Der fork()-Aufruf ermöglicht die Erzeugung einer exakten Kopie des ursprünglichen Prozesses [4]. Der Kindprozess erbt den Zustand des Elternprozesses zum Zeitpunkt des fork()-Aufrufs und setzt seine Ausführung ab diesem Punkt fort. Dies ermöglicht die gleichzeitige Ausführung unterschiedlicher Aufgaben oder Programmabschnitte in unabhängigen Prozessen.

Der fork()-Aufruf bildet eine Grundlage für die Erzeugung paralleler Prozesse in einem Betriebssystem. Durch die gemeinsame Nutzung von Ressourcen und die simultane Ausführung ermöglicht er die Implementierung komplexer Aufgaben und die Nutzung der Rechenleistung von Mehrkernprozessoren.

```
// Start the controller process
while ((controllerPID = fork()) < 0) {
  fprintf(stderr, "Error creating the controller process. Re-try...\n");
  sleep(1);
}

if (controllerPID == 0) {
  // controller-process
  controllerProcess(messageQueueID, getpid());
  exit(0);
}

// Start the watchdog process
watchdogPID = getpid();
watchdogProcess(messageQueueID, watchdogPID, controllerPID);
```

Listing 6: Erzeugung Watchdog- und Controller-Prozesse

Der gegebene Codeabschnitt initiiert das Prozessmanagementsystem, bestehend aus einem Controller-Prozess und einem Watchdog-Prozess. Der Controller-Prozess wird gestartet, in dem eine Schleife verwendet wird, um den Fork-Vorgang auszuführen. Falls der Fork-Vorgang

einen negativen Rückgabewert liefert, wird eine Fehlermeldung ausgegeben und nach einer Sekunde ein erneuter Versuch unternommen. Wenn der Fork-Vorgang erfolgreich ist und einen Wert von 0 zurückgibt, wird der Kindprozess erzeugt. In diesem Kindprozess wird die Funktion controllerProcess() aufgerufen und dabei die Nachrichtenwarteschlangen-ID sowie die Prozess-ID des Controllers übergeben.

4 Implementierung des Steuerprozesses

4.1 Beschreibung des Steuerprozesses

Die Funktion controllerProcess() implementiert einen Controller-Prozess, der periodisch Fehler simuliert und entsprechende Maßnahmen ergreift. Diese Fehler werden dann von der Funktion watchdogProcess() erkannt und entsprechend behandelt. Für das Projekt wurden die folgenden drei Strategien ausgewählt: das Erfassen eines Timeout-Fehlers, das Erkennen eines Temperaturfehlers und die Behandlung eines Deadlock-Fehlers.

```
void controllerProcess(int messageQueueID, pid_t controllerPID) {

    errorDetected.errorDetected = false;

    while (true) {

        int selectedError;

        switch (selectedError) {

#if START_TIMEOUT_ERROR
            case ERROR_OPTION_TIMEOUT:

                createTimeoutError(messageQueueID, controllerPID);

                break;
#endif
#if START_TEMPERATURE_ERROR
            case ERROR_OPTION_TEMPERATURE:

                createTemperatureError(messageQueueID, controllerPID);

                break;
#endif
#if START_DEADLOCK
            case ERROR_OPTION_DEADLOCK:

                createDeadlock();

                break;
#endif

        }
        sleep(2);
    }
 }
```

Listing 7: Implementierung des Controller-Prozesses

4.1.1 Umsetzung der Timeout-Error-Funktion

Wenn es zu einer Zeitüberschreitung kommt, ruft der Controller die Funktion createTimeoutError(messageQueueID, controllerPID) auf, um eine mögliche Zeitüberschreitung zu erstellen und zu simulieren.

```
void createTimeoutError(int messageQueueID, pid_t controllerPID) {

    time_t startTime = time(NULL);
    printf("check if there any Timeout in the loop\n");
    errorDetected.errorDetected = false;
    while ((time(NULL) - startTime) < LOOP_DURATION) {

        int randomDelay = rand() % delay;
        sleep(randomDelay);
        if ((time(NULL) - startTime) >= LOOP_DURATION) {
            errorDetected.errorDetected = true;
            printf(RED_COLOR"Controller process: Timeout detected in the
loop! Taking appropriate actions..."RESET_COLOR"\n");
            Error error = {true, TIMEOUT_ERROR};
            sendWatchdogMessage(messageQueueID, error, controllerPID);
        }
        sleep(1);

    }
    if (!errorDetected.errorDetected) {
        printf("No timeout detected\n");
        startTime = 0;
    }
    sleep(2);

}
```

Listing 8 :Implementierung der Timeout-Fehlerfunktion

Die Funktion erstellt einen Timeout-Fehler, indem sie eine Schleife ausführt, die für eine bestimmte Zeitdauer (definiert durch die Konstante LOOP_DURATION) läuft. Innerhalb der Schleife wird zunächst eine zufällige Verzögerung generiert und gewartet, bevor überprüft wird, ob die Zeitdauer für die Schleife überschritten wurde. Wenn dies der Fall ist, wird der Fehler erkannt und entsprechende Aktionen werden unternommen. In diesem Fall wird die Funktion sendWatchdogMessage() aufgerufen, um eine Nachricht an den Watchdog-Prozess zu senden.

4.1.2 Implementierung der Deadlock-Error-Funktion

Deadlocks stellen problematische Situationen in Computersystemen dar, bei denen zwei oder mehr Prozesse blockiert sind und keiner von ihnen voranschreiten kann [7]. Jeder Prozess befindet sich in der Warteschleife für eine Ressource, die von einem anderen Prozess gehalten wird, während er gleichzeitig eine Ressource besitzt, auf die der andere Prozess wartet. Solche Deadlocks führen zu einem Stillstand im System und können den normalen Ablauf der Prozesse stören. Die Vermeidung von Deadlocks ist von essenzieller Bedeutung, da sie zu einer Blockade des Systems führen und den reibungslosen Betrieb beeinträchtigen können.

Tritt ein Deadlock auf, kann das gesamte System inaktiv werden und auf externe Eingaben oder Unterbrechungen nicht mehr reagieren. Diese Situation kann zu Systemausfällen, Datenverlust oder anderen unerwünschten Folgen führen. Zur Veranschaulichung dieser Problematik wird im Controller mithilfe von Semaphoren ein Deadlock erzeugt. Der Semaphore spielt dabei eine bedeutende Rolle bei der Verwaltung der Ressourcen während eines Deadlocks. Er unterstützt die Koordination des Zugriffs auf kritische Abschnitte des Codes, indem er die Synchronisierung zwischen den beteiligten Prozessen gewährleistet.

Durch die Verwendung des Semaphors wird sichergestellt, dass ein Prozess eine Ressource erst dann freigibt, wenn er sicher ist, dass der andere Prozess sie nicht mehr benötigt. Dadurch wird verhindert, dass beide Prozesse gleichzeitig auf die Ressourcen warten und somit ein Deadlock entsteht. Zur Vereinfachung des Programms wird der Semaphore absichtlich mit einem anfänglichen Wert von 0 initialisiert. Dadurch befindet sich der Semaphore sofort in einem blockierten Zustand. In diesem Zustand stehen keine Ressourcen zur Verfügung, und jeder Versuch, den Semaphore zu sperren, führt sofort zu einer Blockade. Dies dient dazu, den Controller-Prozess zu blockieren und die Schaffung eines Deadlocks zu veranschaulichen, wenn die Funktion createDeadlock() vom Controller aufgerufen wird.

```
void createDeadlock() {

    printf(RED_COLOR" Deadlock.\n"RESET_COLOR);
    lock_semaphore(sem_id, 0);
}
```

Listing 9: Erzeugung eines Deadlocks

4.1.3 Entwicklung der Temperatur-Error-Funktion

Bei einer auftretenden Überhitzung initiiert der Controller den Aufruf der Funktion createTimeoutError() um eine mögliche Zeitüberschreitung zu erzeugen und zu simulieren. Die vorliegende Funktion createTemperatureError() erzeugt eine zufällige Temperatur im Bereich von 80 bis 100 Grad Celsius. Anschließend überprüft der Code, ob die generierte Temperatur 90 Grad Celsius oder höher beträgt. Falls dies der Fall ist, wird die Funktion sendWatchdogMessage() aufgerufen, um eine Nachricht an den Watchdog-Prozess zu senden.

```
void createTemperatureError(int messageQueueID, pid_t controllerPID) {
    errorDetected.errorDetected = false;
    srand(time(NULL));
    int temperature = rand() % 21 + TEMPERATURE;
    printf(GREEN_COLOR"Current temperature: %d°C\n"RESET_COLOR,
temperature);
    sleep(1);

    if (temperature >= OverheatingDetected) {

        printf(RED_COLOR" Overheating detected, Taking appropriate ac-
tions..."RESET_COLOR"\n");
        Error error = {true, OVERHEATING_ERROR};
        sendWatchdogMessage(messageQueueID, error, controllerPID);
    } else {

        printf("No overheating detected. Everything is normal.\n");
    }
    sleep(2);
}
```

Listing 10: Implementierung der Temperaturfehler-Funktion

5 Implementierung des Watchdog-Prozesses

Die Implementierung des Watchdog-Prozesses stellt einen wesentlichen Bestandteil des Gesamtsystems dar. Nachdem alle potenziellen Fehler-Szenarien im Controller-Prozess umgesetzt wurden, wird der Watchdog-Prozess eingesetzt, um sämtliche Fehlermeldungen vom Controller zu empfangen und entsprechend zu reagieren, indem er versucht, diese Fehler zu beheben. Der Watchdog-Prozess übernimmt somit die Verantwortung für die Überwachung und Aufrechterhaltung der fehlerfreien Funktionalität des Systems.

```
void watchdogProcess(int messageQueueID, pid_t watchdogPID, pid_t
controllerPID) {

    if (ENABLE_PRINT_SYMBOL) { printSymbol(LENGTH_SYMBOL_SIZE); }
    printf(DARK_GREEN_COLOR"The watchdog is active with PID: %d"
RESET_COLOR"\n", watchdogPID);
    setNewControllerPID = false;
    watchdogStarted = true;
    // Get the current time
    time_t startTime = time(NULL);
    while (true) {
            Message message;
            if (setNewControllerPID) { controllerPID = newControllerPID;
}
            // Get the current time
            time_t currentTime = time(NULL);

            // Calculate the time elapsed since the start
            double elapsedTime = difftime(currentTime, startTime);

            if (elapsedTime > MAX_TIME_THRESHOLD) {
                printf(BLUE_COLOR"\nwatchdog process : Expected response
time from controller: %d seconds, current time: %.2f seconds\n"
RESET_COLOR,
                       MAX_TIME_THRESHOLD, elapsedTime);
                startTime = currentTime; // Reset the start time
                restoreSafeState(messageQueueID, controllerPID);
            }

            if (msgrcv(messageQueueID, &message, sizeof(Error), 1,
IPC_NOWAIT) != -1) {


 if (elapsedTime < MIN_TIME_THRESHOLD) {
                    printf(PREFIX_WATCHDOG " Early response from
controller: %.2f seconds,Expected response time %d seconds.\n",
elapsedTime,MIN_TIME_THRESHOLD);
                    startTime = currentTime; // Reset the start time
                    restoreSafeState(messageQueueID, controllerPID);

                }
```

```
                printf(YELLOW_COLOR"The Watchdog process  with PID  %d
received a message from the controller with PID:  %d and with error:
%s\n"RESET_COLOR, watchdogPID, controllerPID, toString(
message.error.errorCode));
                fixError(messageQueueID, message.error, controllerPID);

                message.messageType = 2;
                message.error.errorDetected = false;
                message.error.errorCode = ERROR_NO_ERROR;
                msgsnd(messageQueueID, &message, sizeof(Error), 0);

                printf(YELLOW_COLOR"Watchdog sent a message to the con-
troller: %s\n"RESET_COLOR,
                       toString(message.error.errorCode));
                if (ENABLE_PRINT_SYMBOL) {
                   printSymbol(LENGTH_SYMBOL_SIZE); }
                if (!message.error.errorDetected) {
                    printf(BLUE_COLOR"Error fixed. Proceeding with the
controller process.\n"RESET_COLOR);
                    if (ENABLE_PRINT_SYMBOL)
{ printSymbol(LENGTH_SYMBOL_SIZE); }
                    sleep(1);
                }
                startTime = currentTime;

            }
            usleep(PERIOD * 1000);
        }
    }
```

Listing 11: Implementierung des Watchdog-Prozesses

In der Schleife des Watchdog-Prozesses wird die Funktion msgrcv verwendet, um Nachrichten aus der Warteschlange zu empfangen. Diese Nachrichten werden in einer Variablen namens message gespeichert und als Instanz der Struktur Error erwartet. Nach dem Empfang ruft der Watchdog-Prozess die Funktion fixError() auf, um die vorliegenden Probleme zu lösen. Dabei werden die messageQueueID, der aus der empfangenen Nachricht abgeleitete Fehler und die Prozess-ID des Controller-Prozesses (controllerPID) als Parameter übergeben. Sobald das Problem erfolgreich behoben wurde, sendet der Watchdog-Prozess eine Nachricht an den Controller-Prozess, um den aktuellen Status der Fehlerbehebung zu kommunizieren. Die Funktion fixError() implementiert eine strukturierte Entscheidungsstruktur in Form einer Switch-Anweisung. Diese Anweisung ermöglicht es, basierend auf der Art des aufgetretenen Fehlers, spezifische Unterfunktionen aufzurufen, um eine angemessene Reaktion auf den Fehler zu ermöglichen und die entsprechende Fehlerbehebung durchzuführen. Das primäre Ziel dieser Funktion

besteht darin, die erforderlichen Funktionen aufzurufen, um den Controller-Prozess in einen sicheren und funktionsfähigen Zustand zurückzuführen.

```
void fixError(int messageQueueID, Error error, pid_t controllerPID) {
    if (!error.errorDetected)
        return;

    printf(YELLOW_COLOR"The watchdog process is fixing the problem: %s
\n"RESET_COLOR, toString(error.errorCode));
    switch (error.errorCode) {
        case TIMEOUT_ERROR:
            handleWatchdogTimeoutStrategy(messageQueueID, controllerPID);
            break;
        case OVERHEATING_ERROR:
            handleWatchdogTemperatureStrate-gy(messageQueueID,
controllerPID);
            break;
        case DEADLOCK_ERROR:
            handelDeadlockStrategy(messageQueueID, controllerPID);
            break;
        case UNKNOWN_ERROR:
            restoreSafeState(messageQueueID, controllerPID);
            break;
        default:
            printf("Controller: No errors detected. Proceeding...\n");
    }
}
```

Listing 12: Implementierung der Fehlerbehebungsfunktion

5.1 Handhabung der Timeout-Strategie

Eine mögliche Lösungsstrategie für den Fall einer Zeitüberschreitung besteht darin, das gesamte System neu zu starten. In unserem spezifischen Kontext wird der Watchdog den Controller-Prozess beenden und einen neuen Prozess erzeugen. Sobald eine Zeitüberschreitung erkannt und vom WatchdogProcess() registriert wird, erfolgt die Aktivierung der Funktion restoreSafeState(). Der primäre Zweck dieser Funktion besteht darin, den Controller-Prozess in einen sicheren Zustand zurückzuversetzen.

```
void restoreSafeState(int messageQueueID, pid_t controllerPID) {

    printf(YELLOW_COLOR"Watchdog process: Restoring the safe
state...\n"RESET_COLOR);
    printf(YELLOW_COLOR"Watchdog process:"RED_COLOR " Restarting the
program..." RESET_COLOR "\n");
    if (ENABLE_PRINT_SYMBOL) { printSymbol(LENGTH_SYMBOL_SIZE); }
```

```
    printf(RED_COLOR"Watchdog process killed the controller process with
PID: %d\n"RESET_COLOR, controllerPID);
    // Kill the controller process
    if (kill(controllerPID, SIGKILL) == 0) {
        printf(RED_COLOR "Controller process with PID %d has been
successfully killed.\n" RESET_COLOR, controllerPID);
    } else {
        printf(RED_COLOR "Failed to kill controller process with PID
%d.\n" RESET_COLOR, controllerPID);
    }
    // Wait for the controller process to exit and clean up its resources
    int status;
    pid_t result;
    do {
        result = waitpid(controllerPID, &status, 0);
    } while (result == -1 && errno == EINTR);  // handle interrupt errors

    if (result != -1) {
        if (WIFEXITED(status)) {
            printf(RED_COLOR "Controller process with PID %d has exited
with status: %d\n" RESET_COLOR, controllerPID,
                   WEXITSTATUS(status));
        } else if (WIFSIGNALED(status)) {
            printf(RED_COLOR "Controller process with PID %d was
terminated by signal: %d\n" RESET_COLOR, controllerPID,
                   WTERMSIG(status));
        }
    } else {

        printf(RED_COLOR "Failed to wait for the controller process with
PID %d\n" RESET_COLOR, controllerPID);
    }

    // Start the new controller process
    newControllerPID = fork();
    if (newControllerPID == -1) {
        printf("Error creating new controller process.\n");
        exit(1);
    } else if (newControllerPID == 0) {
        controllerProcess(messageQueueID, getpid());
        exit(0);
    } else {
        printf(RED_COLOR"Watchdog process continues with the same PID:
%d\n"RESET_COLOR, getpid());
        setNewControllerPID = true;
        isCoolingSystemActivated = false;

    }
}
```

Listing 13: Wiederherstellung des sicheren Zustands

Zu Beginn wird der Controller-Prozess durch den Einsatz der Funktion kill() in Verbindung mit dem Signal SIGKILL terminiert. Im Falle eines erfolgreichen Abschlusses dieses Vorgangs erfolgt eine entsprechende Benachrichtigung. Tritt jedoch ein Fehler auf, wird eine Fehlermeldung ausgegeben. Anschließend erfolgt das Warten auf das vollständige Beenden des Controller-Prozesses mittels der Funktion waitpid() . Diese Schleife wird solange durchlaufen, bis der Prozess erfolgreich beendet wurde und kein Unterbrechungssignal auftritt. Nach Erfüllung dieser Bedingungen wird ein neuer Controller-Prozess initiiert, indem die Funktion fork() aufgerufen wird, um einen neuen Prozess zu erzeugen. Im Falle eines auftretenden Fehlers wird eine Fehlermeldung erzeugt und es wird eine Sekunde gewartet, bevor ein erneuter Versuch unternommen wird. Falls die Erzeugung des neuen Prozesses erfolgreich ist (erkennbar an der Rückgabe des Wertes 0), wird die Funktion controllerProcess() aufgerufen. Dabei werden die ID der Nachrichtenwarteschlange sowie die aktuelle Prozess-ID als Parameter übergeben.

5.2 Umsetzung der Deadlock-Strategie

Der Watchdog ermittelt nach einem bestimmten Zeitintervall die Inaktivität des controllerProcess() durch das Einstellen von Nachrichtenübermittlungen. Dies führt zur Ausführung der Funktion restoreSafeState() die den Controller in einen sicheren Betriebszustand zurückversetzt.

5.3 Implementierung der Temperatur-Strategie

Sobald eine Überhitzung erkannt wird, greift der Watchdog auf die Funktion activateCooling-System() zurück. Diese Funktion beinhaltet eine maßgeschneiderte Strategie, um das Kühlsystem zu aktivieren. Das Hauptziel besteht darin, eine effiziente Kühlung des Systems zu gewährleisten.

```
void activateCoolingSystem(int messageQueueID, pid_t controllerPID) {

    if (!isCoolingSystemActivated) {
        printf(RED_COLOR "Cooling system activated."RESET_COLOR "\n");
        isCoolingSystemActivated = true;
    } else {
        printf(RED_COLOR"Cooling system is already activated."RESET_COLOR
"\n");
        printf(RED_COLOR"the controller must be restarted"
RESET_COLOR"\n");
        sleep(2);
        restoreSafeState(messageQueueID, controllerPID);
    }
}
```

Listing 14: Implementierung der Aktivierung des Kühlsystems

6 Testausführung und Verhaltensbeobachtung des individuellen Watchdog-Mechanismus

Nach erfolgreicher Implementierung beider Prozesse ist es nun möglich, die erzielten Resultate zu beobachten und zu analysieren.

6.1 Deadlock-Szenario

In diesem Szenario steht der Fokus auf der Darstellung eines Deadlocks. Der gewählte Fehler simuliert einen Zustand, in dem der Prozess ControllerProcess() blockiert ist und keine Möglichkeit zur Fortsetzung hat. Der Watchdog erwartet eine Antwort innerhalb von 20 Sekunden. Wenn diese nicht eintritt, weil der Controller blockiert ist und keine weiteren Aktionen ausführen kann, greift der Watchdog ein. Er versetzt den Controller in einen sicheren Zustand, generiert einen neuen Controller und beendet den blockierten Controller.

```
**************************************************************************
  Watchdog: is active with PID: 6491
Controller: is active with PID: 6492
**************************************************************************
**************************
Which error do you want to simulate?
1. Timeout error
2. Temperature error
3. Deadlock
Your selection: 3
Controller: is currently simulating whether errors exist.
Controller: Deadlock.
  Watchdog: Expected response time from controller: 20 seconds, current
time: 21.00 seconds
  Watchdog: Restoring the safe state...
  Watchdog: Restarting the program...
**************************************************************************
  Watchdog: killed the controller process with PID: 6492
  Watchdog: Controller process with PID 6492 has been successfully
killed.
  Watchdog: Controller process with PID 6492 was terminated by signal: 9
  Watchdog: continues with the same PID: 6491
Controller: is active with PID: 6496
**************************************************************************
```

Listing 15:Ausgabe des Deadlock-Szenarios

6.2 Timeout-Szenario

In diesem Szenario erfolgt die Erkennung einer Zeitüberschreitung innerhalb des Controller-Process(). Daraufhin wird eine Mitteilung an das WatchdogProcess() gesandt. Der Watchdog erkennt den Fehler, ruft die restoreSafeState() auf und versetzt den Controller in einen gesicherten Zustand zurück.

```
**************************************************************************

  Watchdog: is active with PID: 6602
Controller: is active with PID: 6603
**************************************************************************
Which error do you want to simulate?
1. Timeout error
2. Temperature error
3. Deadlock
Your selection: 1

Controller: is currently simulating whether errors exist.
Controller: check if there any Timeout in the loop
Controller: Timeout detected in the loop! Taking appropriate actions...
Controller: The message has been sent to the watchdog: Error Timeout
**************************************************************************
  Watchdog: with PID  6602 received a message from the controller with
PID:  6603 and with error: Error Timeout
  Watchdog: is fixing the problem: Error Timeout
  Watchdog: Restoring the safe state...
  Watchdog: Restarting the program...
**************************************************************************
  Watchdog: killed the controller process with PID: 6603
  Watchdog: Controller process with PID 6603 has been successfully
killed.
  Watchdog: Controller process with PID 6603 was terminated by signal: 9
  Watchdog: continues with the same PID: 6602
  Watchdog: sent a message to the controller: Problem Solved
**************************************************************************
  Watchdog: Error fixed. Proceeding with the controller process.
**************************************************************************
Controller: is active with PID: 6604
**************************************************************************
```

Listing 16:Ausgabe des Timeout-Szenarios

6.3 Temperature-Szenario

Im Temperature-Szenario wird beim ersten Auftreten einer Überhitzung im ControllerProcess() eine Nachricht an den WatchdogProcess() gesendet, um das Kühlsystem zu aktivieren. Beim zweiten Auftreten einer Überhitzung wird die Funktion RestoreSafeState() aufgerufen,

wobei das Kühlsystem bereits aktiv ist. Der Watchdog erkennt den Zustand und initiiert die notwendigen Maßnahmen. Dieses Verhalten ähnelt dem im Timeout-Szenario beschriebenen Prozess, wobei der Watchdog den Controller in einen sicheren Zustand versetzt, einen neuen Controller generiert und den blockierten Controller beendet.

```
**************************************************************************
  Watchdog: is active with PID: 6607
Controller: is active with PID: 6608
**************************************************************************
Which error do you want to simulate?
1. Timeout error
2. Temperature error
3. Deadlock
Your selection: 2
Controller: is currently simulating whether errors exist.
Controller: Current temperature: 99°C
Controller: Overheating detected, Taking appropriate actions...
Controller: The message has been sent to the watchdog: Error Overheating
**************************************************************************
  Watchdog: with PID  6607 received a message from the controller with
PID:  6608 and with error: Error Overheating
  Watchdog: is fixing the problem: Error Overheating
  Watchdog: Cooling system activated.
  Watchdog: sent a message to the controller: Problem Solved
**************************************************************************
  Watchdog: Error fixed. Proceeding with the controller process.
**************************************************************************
Controller: is active with PID: 6608
**************************************************************************
Which error do you want to simulate?
1. Timeout error
2. Temperature error
3. Deadlock
Your selection: 2
Controller: is currently simulating whether errors exist.
Controller: Current temperature: 99°C
Controller: Overheating detected, Taking appropriate actions...
Controller: The message has been sent to the watchdog: Error Overheating
**************************************************************************
  Watchdog: with PID  6607 received a message from the controller with
PID:  6608 and with error: Error Overheating
  Watchdog: is fixing the problem: Error Overheating
  Watchdog: Cooling system is already activated.
  Watchdog: the controller must be restarted
  Watchdog: Restoring the safe state...
  Watchdog: Restarting the program...
**************************************************************************
  Watchdog: killed the controller process with PID: 6608
  Watchdog: Controller process with PID 6608 has been successfully
killed.
  Watchdog: Controller process with PID 6608 was terminated by signal: 9
  Watchdog: continues with the same PID: 6607
  Watchdog: sent a message to the controller: Problem Solved
**************************************************************************
```

```
  Watchdog: Error fixed. Proceeding with the controller process.
**************************************************************************
Controller: is active with PID: 6610
**************************************************************************
```

Listing 17: Ausgabe des Temperatur-Szenarios

7 Kurzzeitige Programm-Ausführung und Beobachtung des Verhaltens der Watchdog-Mechanismen

Nach Ausführung des Programms könnte eine mögliche Ausgabe für einen kurzen Zeitraum folgendermaßen aussehen. Diese Ausgabe umfasst alle implementierten Szenarien, auf die der Watchdog reagiert:

```
**************************************************************************
  Watchdog: is active with PID: 6430
Controller: is active with PID: 6431
**************************************************************************
Controller: is currently simulating whether errors exist.
Controller: check if there any Timeout in the loop
Controller: No timeout detected
Controller: is active with PID: 6431
**************************************************************************
Controller: is currently simulating whether errors exist.
Controller: Deadlock.
  Watchdog: Expected response time from controller: 20 seconds, current
time: 21.00 seconds
  Watchdog: Restoring the safe state...
  Watchdog: Restarting the program...
**************************************************************************
  Watchdog: killed the controller process with PID: 6431
  Watchdog: Controller process with PID 6431 has been successfully
killed.
  Watchdog: Controller process with PID 6431 was terminated by signal: 9
  Watchdog: continues with the same PID: 6430
Controller: is active with PID: 6433
**************************************************************************
Controller: is currently simulating whether errors exist.
Controller: Current temperature: 84°C
Controller: Overheating detected, Taking appropriate actions...
Controller: The message has been sent to the watchdog: Error Overheating
**************************************************************************
  Watchdog: with PID  6430 received a message from the controller with
PID:  6433 and with error: Error Overheating
  Watchdog: is fixing the problem: Error Overheating
  Watchdog: Cooling system activated.
  Watchdog: sent a message to the controller: Problem Solved
**************************************************************************
  Watchdog: Error fixed. Proceeding with the controller process.
**************************************************************************
Controller: is active with PID: 6433
**************************************************************************
Controller: is currently simulating whether errors exist.
Controller: Current temperature: 84°C
Controller: Overheating detected, Taking appropriate actions...
Controller: The message has been sent to the watchdog: Error Overheating
**************************************************************************
  Watchdog: with PID  6430 received a message from the controller with
PID:  6433 and with error: Error Overheating
  Watchdog: is fixing the problem: Error Overheating
  Watchdog: Cooling system is already activated.
```

```
  Watchdog: the controller must be restarted
  Watchdog: Restoring the safe state...
  Watchdog: Restarting the program...
**************************************************************************
  Watchdog: killed the controller process with PID: 6433
  Watchdog: Controller process with PID 6433 has been successfully
killed.
  Watchdog: Controller process with PID 6433 was terminated by signal: 9
  Watchdog: continues with the same PID: 6430
  Watchdog: sent a message to the controller: Problem Solved
**************************************************************************
  Watchdog: Error fixed. Proceeding with the controller process.
**************************************************************************
Controller: is active with PID: 6434
**************************************************************************
Controller: is currently simulating whether errors exist.
Controller: check if there any Timeout in the loop
Controller: Timeout detected in the loop! Taking appropriate actions...
Controller: The message has been sent to the watchdog: Error Timeout
**************************************************************************
  Watchdog: with PID  6430 received a message from the controller with
PID:  6434 and with error: Error Timeout
  Watchdog: is fixing the problem: Error Timeout
  Watchdog: Restoring the safe state...
  Watchdog: Restarting the program...
**************************************************************************
  Watchdog: killed the controller process with PID: 6434
  Watchdog: Controller process with PID 6434 has been successfully
killed.
  Watchdog: Controller process with PID 6434 was terminated by signal: 9
  Watchdog: continues with the same PID: 6430
  Watchdog: sent a message to the controller: Problem Solved
**************************************************************************
  Watchdog: Error fixed. Proceeding with the controller process.
**************************************************************************
Controller: is active with PID: 6435
**************************************************************************
Controller: is currently simulating whether errors exist.
Controller: Deadlock.
  Watchdog: Expected response time from controller: 20 seconds, current
time: 21.00 seconds
  Watchdog: Restoring the safe state...
  Watchdog: Restarting the program...
**************************************************************************
  Watchdog: killed the controller process with PID: 6435
  Watchdog: Controller process with PID 6435 has been successfully
killed.
  Watchdog: Controller process with PID 6435 was terminated by signal: 9
  Watchdog: continues with the same PID: 6430
Controller: is active with PID: 6436
**************************************************************************
Controller: is currently simulating whether errors exist.
Controller: check if there any Timeout in the loop
Controller: Timeout detected in the loop! Taking appropriate actions...
Controller: The message has been sent to the watchdog: Error Timeout
**************************************************************************
  Watchdog: with PID  6430 received a message from the controller with
PID:  6436 and with error: Error Timeout
  Watchdog: is fixing the problem: Error Timeout
  Watchdog: Restoring the safe state...
  Watchdog: Restarting the program...
**************************************************************************
```

```
  Watchdog: killed the controller process with PID: 6436
  Watchdog: Controller process with PID 6436 has been successfully
killed.
  Watchdog: Controller process with PID 6436 was terminated by signal: 9
  Watchdog: continues with the same PID: 6430
  Watchdog: sent a message to the controller: Problem Solved
**************************************************************************
  Watchdog: Error fixed. Proceeding with the controller process.
**************************************************************************
Controller: is active with PID: 6438
```

Listing 18 Kurzzeitige Ausgabe während Programm-Ausführung

Es könnte die Möglichkeit bestehen, dass eine Nachricht vor dem erwarteten Zeitpunkt eintrifft. In einer solchen Situation erkennt der Watchdog den Fehler und ruft die Funktion restoreSafeState() auf, um den Controller in einen sicheren Zustand zurückzusetzen.

```
Controller: is active with PID: 5855
**************************************************************************
Controller: is currently simulating whether errors exist.
Controller: Current temperature: 93°C
Controller: Overheating detected, Taking appropriate actions...
  Watchdog: Early response from controller: 1.00 seconds, Expected
response time 2 seconds.
  Watchdog: Restoring the safe state...
  Watchdog: Restarting the program...
  Watchdog: killed the controller process with PID: 5855
  Watchdog: Controller process with PID 5855 has been successfully
killed.
  Watchdog: Controller process with PID 5855 was terminated by signal: 9
  Watchdog: continues with the same PID: 5850
Controller: is active with PID: 5856
```

Listing 19: Ausgabe einer früheren Nachricht

8 Fazit zur Arbeit

Diese Arbeit hat sich ausführlich mit Watchdog-Mechanismen und ihrer Implementierung in einer Entwicklungs- und Testumgebung auseinandergesetzt. Die Einleitung erläuterte den Hintergrund und die Bedeutung dieser Mechanismen und formulierte klare Forschungsziele. Im theoretischen Rahmen wurden die Fundamente von Watchdogs eingehend behandelt, von ihrer Definition und ihrem Zweck bis hin zu den verschiedenen Arten und Funktionen. Besondere Aufmerksamkeit wurde der Integration von Watchdogs in ISO 26262 und AUTOSAR geschenkt, insbesondere im Hinblick auf Health-Monitoring-Funktionen. Die Methodik und Implementierung veranschaulichten die Auswahl einer Entwicklungsplattform und die Festlegung von Kommunikationsmethoden. Es wurden zwei Prozesse entwickelt: der Controller-Prozess und der Watchdog-Prozess. Die Implementierung von Timeout-, Deadlock- und Temperatur-Error-Funktionen im Controller-Prozess sowie die entsprechenden Strategien im Watchdog-Prozess wurden im Detail beschrieben. Die Durchführung von Tests und Verhaltensbeobachtungen führte zu bedeutsamen Erkenntnissen in verschiedenen Szenarien, darunter Deadlock-, Timeout- und Temperature-Szenarien. Diese Tests belegten die Effektivität der implementierten Watchdog-Mechanismen bei der Erkennung und Behebung von Fehlern.

Im Hinblick auf zukünftige Entwicklungen und Erweiterungen dieser Abschlussarbeit bieten sich vielfältige Möglichkeiten. Die aktuelle Arbeit beschränkte sich auf die Überwachung eines einzelnen Controllers durch einen Watchdog. Eine entscheidende Verbesserungsmöglichkeit besteht darin, die Infrastruktur so weiterzuentwickeln, dass mehrere Controller unabhängig voneinander durch den Watchdog überwacht und konfiguriert werden können. Dies ermöglicht es, jeden Controller individuell an spezifische Anforderungen anzupassen, um besser auf verschiedene Anforderungen und Herausforderungen in der Zukunft reagieren zu können.

Literaturverzeichnis

[1] AHMED, Mazen und Mona SAFAR. Formal Verification of AUTOSAR Watchdog Manager Module Using Symbolic Execution. In: 2018 30th International Conference on Microelectronics (ICM) [online]. IEEE, 2018 [Zugriff am: 28.05.2023]. ISBN 9781538681671. Verfügbar unter: doi:10.1109/icm.2018.8704088

[2] ASCHE, Rüdiger R. Embedded Controller [online]. Wiesbaden: Springer Fachmedien Wiesbaden, 2016 [Zugriff am: 20.05.2023]. ISBN 9783658148492. Verfügbar unter: doi:10.1007/978-3-658-14850-8

[3] EL-BAYOUMI, Abdullah. ISO-26262 Compliant Safety-Critical Autonomous Driving Applications: Real-Time Interference-Aware Multicore Architectures. International Journal of Safety and Security Engineering [online]. 2021, 11(1), 21–34 [Zugriff am: 01.06.2023]. ISSN 2041-904X. Verfügbar unter: doi:10.18280/ijsse.110103

[4] fork() in C - GeeksforGeeks. GeeksforGeeks [online]. [Zugriff am: 26.06.2023]. Verfügbar unter: https://www.geeksforgeeks.org/fork-system-call/

[5] GANSSLE, Jack. Designing Great Watchdog Timers for Embedded Systems. The Ganssle Group [online].[Zugriff am: 22.05.2023]. Verfügbar unter: http://www.ganssle.com/watchdogs.htm

[6] High-Voltage Watchdog Timers Enhance Automotive System Safety. Mixed-signal and digital signal processing ICs | Analog Devices [online].[Zugriff am: 28.07.2023]. Verfügbar unter: https://pdfserv.maximintegrated.com/en/an/AN4210.pdf

[7] HICKS, Gabriel. Processes, Threads, Deadlock, Semaphores, and More. *medium* [online]. 06.04.2021 [Zugriff am: 07.08.2023]. Verfügbar unter: https://medium.com/geekculture/processes-threads-deadlock-semaphores-and-more-f70be5395ef6

[8] IPC using Message Queues - GeeksforGeeks. GeeksforGeeks [online].[Zugriff am: 26.06.2023]. Verfügbar unter: https://www.geeksforgeeks.org/ipc-using-message-queues/

[9] LAMBERSON, Jim. Single and Multistage Watchdog Timers [online]. [Zugriff am: 20.05.2023]. Verfügbar unter: http://www.sensoray.com/downloads/appnote_826_watchdog_1.0.0.pdf

[10] MURPHY, Niall und Michael BARR. Watchdog Timers [online].[Zugriff am: 21.05.2023]. Verfügbar unter: https://webpages.charlotte.edu/~jmconrad/ECGR4101Common/notes/Watchdog%20Timer%20-%20bcorner.pdf

[11] NAKAGAWA, Elisa Yumi und Pablo OLIVEIRA ANTONINO, Hrsg. Reference Architectures for Critical Domains [online]. Cham: Springer International Publishing, 2023 [Zugriff am: 28.05.2023]. ISBN 9783031169564. Verfügbar unter: doi:10.1007/978-3-031-16957-1

[12] Watchdog Stack in AUTOSAR [online]. Embedded Tutor. [Zugriff am 28. Mai 2023]. Verfügbar unter: https://www.embeddedtutor.com/2021/04/watchdog-stack-in-autosar.html

[13] Watchdog Driver in AUTOSAR. *Embedded* Tutor [online]. 25.04.2021 [Zugriff am: 30.07.2023]. Verfügbar unter: https://www.embeddedtutor.com/2021/04/watchdog-driver-in-autosar.html

[14] Q&A Watchdog Timer Configuration for DRV3205-Q1. Analog Embedded processing Semiconductor company, [online]. [Zugriff am: 29.07.2023]. Verfügbar unter: https://www.ti.com/lit/an/slva831/slva831.pdf?ts=1690546469862&ref_url=https%3A%2F%2Fwww.google.com%2F

[15] Requirements on Health Monitoring [online]. [Zugriff am 28. Mai 2023]. Verfügbar unter: https://www.autosar.org/fileadmin/standards/R2111/FO/AUTOSAR_RS_HealthMonitoring.pdf

[16] ROSENSTATTER, Thomas und Tomas OLOVSSON. Towards a Standardized Mapping from Automotive Security Levels to Security Mechanisms. In: 2018 21st International Conference on Intelligent Transportation Systems (ITSC) [online]. IEEE, 2018 [Zugriff am: 28.07.2023]. ISBN 9781728103211. Verfügbar unter: doi:10.1109/itsc.2018.8569679

[17] THORNTON, SCOTT. RTOS vs Standard operating system and how to choose an RTOS. Tips on coding, designing, and embedding with microcontrollers [online]. 04.06.2021 [Zugriff am: 07.08.2023]. Verfügbar unter: https://www.microcontrollertips.com/real-time-standard-how-to-choose-rtos/

[18] ZIMMERMANN, Werner und Ralf SCHMIDGALL. Bussysteme in der Fahrzeugtechnik [online]. Wiesbaden: Springer Fachmedien Wiesbaden, 2014 [Zugriff am: 27.05.2023]. ISBN 9783658024185. Verfügbar unter: doi:10.1007/978-3-658-02419-2