Implementierung einer interruptgesteuerten Benutzerschnittstelle

auf einem Low-Power-Mikrocontroller

Bachelor-Thesis

zur Erlangung des akademischen Grades
Bachelor of Science (B. Sc.) im Studienfach Angewandte Informatik

Hochschule Konstanz
Technik, Wirtschaft und Gestaltung

vorgelegt von:	Julian Rapp
Matrikelnummer:	304875
Erstgutachter:	Prof. Dr. Irenäus Schoppa
Zweitgutachter:	Prof. Dr. Heiko von Drachenfels
eingereicht in:	Konstanz, am 30. Juni 2025

Zitat

So I listen to the radio and all the songs we used to know
So I listen to the radio remember where we used to go

(The Corrs – Radio)

Abstract

Die vorliegende wissenschaftliche Arbeit behandelt die Planung und Entwicklung einer interruptgesteuerten Benutzerschnittstelle zur Überwachung und Manipulation von Registern und Speicherzellen. Die Entwicklung soll auf Basis eines Low-Power-Microcontrollers der MSP430 Familie von Texas Instruments erfolgen.
Es erfolgt eine umfangreiche Planungs-/ und Entwicklungsphase, die in Anlehnung an den Rational Unified Process dokumentiert wird. Auf Basis dieser Planung wird die Software dann als austauschbares Modul implementiert. Der Vorgang der praktischen Umsetzung wird schriftlich dokumentiert. Zum Abschluss der Arbeit wird die Erweiterung im Live-Betrieb des Praktikums von Microprozessorsysteme zum Einsatz kommen.

Inhaltsverzeichnis

Abkürzungsverzeichnis und Glossar

(Opcode) ..... Auch op code oder operation code, ist eine meist in hexadezimaler Schreibweise angegebene Zahl, die die Nummer eines Maschinenbefehls für einen bestimmten Prozessortyp angibt.

16-Bit-RISC-Microcontroller Mikrocontroller mit 16-Bit-Registerbreite und reduzierter Befehlssatzarchitektur (Reduced Instruction Set Computer)

Auxiliary Clock (ACLK) Niederfrequente Taktquelle in Mikrocontroller-Systemen, die typischerweise von einem Quarzoszillator gespeist wird und für energiesparende Betriebsmodi verwendet wird.

Breakpoints ..... Bezeichnet in der Softwareentwicklung eine vom Entwickler bewusst gesetzte Unterbrechung im Programmablauf, die typischerweise zur Laufzeit-Debugging-Zwecken verwendet wird. Beim Erreichen dieses Punkts wird die Ausführung des Programms angehalten, sodass der aktuelle Zustand (z. B. Variableninhalte, Stack, Speicher) analysiert werden kann.

Clock-Source ..... Eine Referenz auf ein periodisches Zeitsignal um zeitliche Abläufe zu synchronisieren; typischerweise in Form von Quarzoszillatoren oder externen Taktsignalen.

Compiler-Intrinsics Compiler-spezifische, vordefinierte Funktionen, die direkt in optimierten Assemblercode umgesetzt werden.

Debugger ..... Ein Werkzeug zur schrittweisen Ausführung und Analyse von Programmen. Es erlaubt das Setzen von Haltepunkten, das überprüfen von Speicherinhalten und das Nachvollziehen von Kontrollflüssen zur Fehlersuche und -behebung.

enhanced Universal Serial Communication Interface Serielle Schnittstelle in Mikrocontrollern von Texas Instruments, die verschiedene Kommunikationsprotokolle (z. B. UART, SPI, I^2C) unterstützt.

eUSCI ..... enhanced Universal Serial Communication Interface

FRAM ..... Ferroelectric Random Access Memory

GPIO ..... General Purpose Input/Output

I^2C ..... Inter-Integrated Circuit

interruptgesteuerten Ein Mechanismus zur ereignisorientierten Unterbrechung des normalen Programmablaufs

ISR ..... Interrupt Service Routine

L ..... [

Low-Power-Mikrocontroller Ein Mikrocontroller, der für energieeffiziente Anwendungen optimiert ist. Typischerweise eingesetzt in batteriebetriebenen Embedded-Systemen

Megaherz ..... Maßeinheit für die Frequenz und entspricht einer Million Schwingungen pro Sekunde (1 MHz = 10^6 Rechenschritte).

MIPS ..... Microprozessorsysteme

Modul ..... Eine funktionale Einheit innerhalb eines größeren Systems, die separat entwickelt und gewartet werden kann

Program Counter (PC) Ein Register, das die Speicheradresse des derzeitigen Befehls enthält.

Pulsweitenmodulation (PWM) Ein Verfahren zur Steuerung der Leistungszufuhr, bei dem die mittlere Ausgangsleistung durch Variieren des Abtastverhältnisses eines Rechtecksignals reguliert wird.

RAM ..... Random Access Memory

Registern ..... Speicherzellen, die flüchtig sind und ihre Inhalte beim Ausschalten verlieren

SPI ..... Serial Peripheral Interface

Spy-Bi-Wire (SBW) Zweidraht-Variante des JTAG-Protokolls, die Pin-Anzahl am Target reduziert und besonders für platzkritische Anwendungen von Vorteil ist.

SRAM ..... Static Random Access Memory

Stack Pointer (SP) Ein Register, das die Speicheradresse des letzten oder ersten Datenelements im Stack speichert.

statischen Arbeitsspeicher Schnellster, flüchtiger Speicher mit geringer Kapazität, bestehend aus Flip-Flops welcher meist direkt in der CPU mit eingebaut ist.

Statusregister (SR) Register für eine Reihe von Flags, die von der arithmetisch-logischen Einheit in Abhängigkeit der zuletzt durchgeführten Rechenoperation gesetzt werden.

Sub-Main Clock (SMCLK) Taktgesteuertes Signal, das typischerweise für Peripheriegeräte verwendet wird und sich aus einer frei wählbaren Taktquelle ableiten lässt.

trial & error ..... Heuristische Methode, bei der durch Versuch und Irrtum eine Lösung gefunden wird.

UART ..... Universal Asynchronous Receiver Transmitter

1. Einleitung

1.1. Das Ziel dieser Arbeit

Diese Bachelor-Thesis befasst sich mit der Entwicklung einer *interruptgesteuerten*[1] Benutzerschnittstelle auf einem *Low-Power-Mikrocontroller*[2], zur Überwachung und Manipulation von *Registern*[3] und Speicherzellen in *RAM* und *FRAM*.

Im Zuge des Arbeitsauftrags wird ein unabhängiges *Modul*[4] entwickelt, welches nach Wunsch aktiviert oder deaktiviert wird.

1.2. Die Umgebung, in der die Arbeit entstand

Die Entwicklung der Software geschah in Absprache mit Herr Prof. Dr. Irenäus Schoppa, welcher ein zusätzliches Tool für Studenten im Praktikum von Microprozessorsysteme benötigt.

Als Entwicklungsbasis kam der in *MIPS* herangezogene MSP430FR5729 von Texas Instruments zum Einsatz, welcher bereits ein ausgereifter und etablierter *L*ow-Power-Mikrocontroller]LP-MCU ist. Viele Technologien, die diesem Prozessor zugrunde liegen, werden in dieser Arbeit wissentlich nicht behandelt, um den Rahmen nicht zu sprengen und sich auf das wesentliche zu konzentrieren.

[1] Ein Mechanismus zur ereignisorientierten Unterbrechung des normalen Programmablaufs
[2] Ein Mikrocontroller, der für energieeffiziente Anwendungen optimiert ist. Typischerweise eingesetzt in batteriebetriebenen Embedded-Systemen
[3] Speicherzellen, die flüchtig sind und ihre Inhalte beim Ausschalten verlieren
[4] Eine funktionale Einheit innerhalb eines größeren Systems, die separat entwickelt und gewartet werden kann

1.3. Der Aufbau dieser Arbeit

Aktueller Wissensstand: Der aktuelle Wissensstand beschreibt, auf welchem Wissensniveau sich der Autor im Moment der Aufnahme der Arbeit befand.

Methoden, Technologien und Herangehensweise: Im Kapitel 2.1 werden die zur Planung und Umsetzung verwendeten Technologien und Methoden erläutert. Die grundlegenden Eigenschaften und der Aufbau des Softwareentwicklungsprozesses werden erklärt. Zudem auch die benötigten Dokumentationen referenziert.

Die Konzeptionierung des Moduls: Dieses Kapitel umfasst die Dokumentation der gesamten Planungsphase des Observer-Moduls. Hier wird eine Übersicht über die bereits vorhandene Lösung geschaffen und anschließend die zur Planung erforderlichen Dokumente angefertigt.

Die Entwicklung des Observers: Dieses Kapitel enthält die Dokumentation der tatsächlichen Programmierung der Software. Hier werden die Voraussetzungen zur Implementation geklärt und der Verlauf der Entwicklung anhand von Beispielen schrittweise abgearbeitet.

Fazit und kritische Bewertung: Im Fazit werden die gemachten Erfahrungen und die Ergebnisse der Planung und Entwicklung abschließend zusammengefasst und kritisch bewertet. Zusätzlich wird ein kleiner Ausblick auf Erweiterungsmöglichkeiten und mögliche Optimierungsschritte unternommen.

1.4. Viele Informationen, wenige Quellen ...

Grundsätzlich war es einfach geeignete Quellen zu den Themen rund um die Technologien zu finden, da – wie bereits erwähnt – die Entwicklungsplattform und der Microcontroller weitesgehend etabliert sind. Allerdings können alle wichtigen Informationen aus erster Hand, von dem Hersteller entnommen werden, weshalb andere Quellen unnötig erscheinen.

2. Hauptteil

2.1. überblick über die Entwicklungsplattform MSP430FR5729

Die Auswahl einer geeigneten Entwicklungsplattform bildet die Grundlage für die erfolgreiche Implementierung und Evaluierung eingebetteter Systeme. Im Rahmen dieser Arbeit dient der Mikrocontroller MSP430FR5729 von Texas Instruments als zentrale Hardwarekomponente, dessen Architektur und Funktionalitäten im Folgenden näher betrachtet werden.

Der MSP430FR5729 ist ein Low-Power *16-Bit-RISC-Microcontroller*[5] von Texas Instruments mit einer Maximalen Taktfrequenz von Acht *Megaherz*[6], wodurch er sich besonders für energieeffiziente Anwendungen im Bereich eingebetteter Systeme eignet.

Der Mikroprozessor besitzt 16 Kilobyte an nicht-flüchtigen FRAM, sowie ein Kilobyte *statischen Arbeitsspeicher*[7] (*SRAM*).

Die Versorgungsspannung beträgt 2 bis 3,6 Volt wobei verschiedene Low-Power-Modi verwendet werden können, um den Stromverbrauch zunehmend zu minimieren. Diese beeinflussen den späteren Umgang mit Timer-Interrupts, weil sie den Energieverbrauch im Wartezustand beeinflussen.

Des Weiteren besitzt der Chip Fünf Interne 16-Bit Timer mit jeweils Sieben **Capture and Compare** Registerblöcken. Diese internen Timer stellen eine zentrale

[5] Mikrocontroller mit 16-Bit-Registerbreite und reduzierter Befehlssatzarchitektur (Reduced Instruction Set Computer)

[6] Maßeinheit für die Frequenz und entspricht einer Million Schwingungen pro Sekunde (1 MHz = 10^6 Rechenschritte).

[7] Schnellster, flüchtiger Speicher mit geringer Kapazität, bestehend aus Flip-Flops welcher meist direkt in der CPU mit eingebaut ist.

Komponente für die Realisierung präziser Zeitsteuer Funktionen und die Generierung von Interrupts dar, welche im Detail im nachfolgenden Kapitel 2.1.1 erläutert werden.

Zur externen Kommunikation können Protokolle wie *UART*, I^2C und *SPI* eingesetzt, welche mit 32 Programmierbaren *GPIO*-Pins angeschlossen werden. Kommunikationsschnittstellen sind für die Interaktion mit der Außenwelt und Peripheriegeräten von hoher Bedeutung. Die weitere Ausarbeitung des *enhanced Universal Serial Communication Interface*[8] (*eUSCI*) hierzu in Kapitel 2.1.3. [Texas Instruments 2017, S. 1, Kap. 1.1]

Abbildung Abbildung 2.1 zeigt ein vollständiges Block Diagramm des Mikroprozessors, welches noch einige weitere Eigenschaften und Funktionen auflistet.

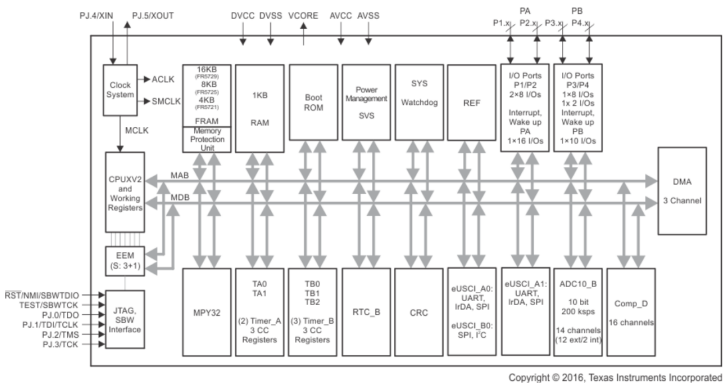

Abb. 2.1.: Block Diagramm MSP430FR5729 Mikrocontroller [Texas Instruments 2017, S. 2, Kap. 1.4]

2.1.1. Timer und Interrupt Service Routinen (ISR)

Timer und Interrupt Service Routinen (*ISR*) stellen fundamentale Bausteine moderner eingebetteter Systeme dar. Sie ermöglichen präzise Zeitsteuerungen, die Reaktion auf externe Ereignisse und somit die Realisierung komplexer, Echtzeitsysteme. Im Folgenden wird die Timer-Architektur des MSP430FR5729 und die zugehörigen ISR-Mechanismen detailliert betrachtet.

[8] Serielle Schnittstelle in Mikrocontrollern von Texas Instruments, die verschiedene Kommunikationsprotokolle (z. B. UART, SPI, I^2C) unterstützt.

Der MSP430FR5729 verfügt über insgesamt fünf 16-Bit-Timer, wobei zwei dem Typ A und drei dem Typ B angehören. Beide Typen ermöglichen vielseitige Zeitsteuerungsfunktionen, weisen jedoch spezifische Unterschiede in ihren Konfigurationsmöglichkeiten auf.

Beide Timer-Typen verfügen über einen gemeinsamen 16-Bit-Zähler sowie bis zu sieben Capture/Compare-Register. Diese Register ermöglichen die Implementierung verschiedenster Funktionen. Die Capture-Funktionalität dient dazu, den aktuellen Zählerwert bei einem externen oder internen Ereignis präzise zu erfassen. Dies ist beispielsweise nützlich für die Messung von Pulsweiten oder Frequenzen. Die Compare-Funktionalität hingegen erlaubt den Vergleich des aktuellen Zählerstandes mit einem in den Compare-Registern hinterlegten Wert. Bei einer übereinstimmung kann eine konfigurierbare Aktion ausgelöst werden, wie beispielsweise das Setzen oder Rücksetzen eines Ausgangspins oder das Generieren eines Interrupts. Die vielseitigen Einstellungsmöglichkeiten dieser Register erlauben die Realisierung komplexer Zeitsteuerungsaufgaben. [Texas Instruments 2018b, S. 333, Kap. 11 & S. 355, Kap. 12], [Davies 2008, S. 287, Kap. 8.3 & S. 194, Kap. 6.8.2]

Die Timer des Typs B weisen im Vergleich zu den Timern des Typs A erweiterte Konfigurationsmöglichkeiten auf. Ein wesentlicher Unterschied zu einem Timer des Typs A besteht in der Konfigurierbarkeit der Timer-Länge auf 8, 10, 12 oder 16 Bit, was eine flexible Anpassung der Zählauflösung und der überlaufperiode für unterschiedliche Auflösungen ermöglicht. Weiterhin sind alle Capture/Compare-Blöcke doppelt gepuffert. Diese doppelte Pufferung erlaubt das Laden neuer Vergleichswerte, während eines aktiven Zählzyklus, wodurch unerwünschte Effekte oder Inkonsistenzen in den Ausgangssignalen vermieden werden. Zudem können alle Ausgänge des Timers B auf einen hochohmigen Zustand umgeschaltet werden, was in bestimmten Applikationen vorteilhaft sein kann. Ein weiterer wichtiger Unterschied besteht darin, dass die Capture/Compare-Eingänge nicht synchronisiert sind und somit asynchron zu dem internen Takt des Timers operieren können, was in bestimmten Szenarien die Erfassung externer Ereignisse erleichtert. [Texas Instruments 2018b, S. 356, Kap. 12.1.1], [Davies 2008, S. 353, Kap. 8.9]

Für die präzise Steuerung und Ereignisbehandlung bieten die Timer verschiedene Betriebsarten, die im Folgenden näher erläutert werden.

2.1.1.1. Timer Zählweisen

Der Zählmodus, bestimmt die interne Zählweise des Timers. Die Timer unterstützen typischerweise mehrere Varianten dieses Modus, um unterschiedlichen Anforderungen gerecht zu werden.

- **Up Mode:** Im Up Mode (Additive Zählweise) beginnt der Zähler bei Null und inkrementiert seinen Wert mit jedem Taktimpuls der gewählten *Clock-Source*[9]. Er erreicht einen vordefinierten Maximalwert, der im Compare-Register gespeichert ist, und beginnt dann wieder von Null zu zählen. Ein überlauf-Interrupt wird generiert, sobald der Zähler den Wert von CCR0 erreicht. Dieser Modus eignet sich ideal für die Erzeugung periodischer Ereignisse oder die Messung von Zeitintervallen bis zu einem bestimmten Grenzwert. Beispielsweise kann durch die Wahl einer geeigneten Clock-Source und eines passenden Wertes im Compare-Register eine präzise Zeitbasis für periodische Aufgaben geschaffen werden. [Texas Instruments 2018b, S. 337, Kap. 11.2.3.1 & S. 359, Kap. 12.2.3.1],[Davies 2008, S. 330, Kap. 8.6]
- **Continuous Mode:** Der Continuous Mode lässt den Zähler von Null bis zum maximal möglichen Wert (FFFFh für 16-Bit-Timer) zählen und anschließend wieder bei Null beginnen. Ein überlauf-Interrupt wird generiert, wenn der Zähler vom Wert von FFFFh auf 0 überläuft. [Texas Instruments 2018b, S. 338, Kap. 11.2.3.2 & S. 360, Kap. 12.2.3.2] Dieser Modus ist besonders nützlich, wenn längere, voneinander unabhängige Zeitintervalle zu messen oder wenn eine freilaufende Zeitbasis benötigt wird, um Ereignisse in Bezug auf den Zählerstand, ohne einen periodischen Neustart durch das Compare-Register, zu erfassen. [Texas Instruments 2018b, S. 338, Kap. 11.2.3.3 & S. 360, Kap. 12.2.3.3], [Davies 2008, S. 318, Kap. 8.5]
- **Up/Down Mode:** Der Up/Down Mode (Auf-/Abwärtszählmodus) kombiniert das Auf- und Abzählen. Der Zähler beginnt bei Null, zählt Zyklisch bis zum festgelegten Wert im Compare-Register und dann wieder bis Null herunter. Ein überlauf-Interrupt wird generiert, wenn der Zähler den Wert von CCR0 erreicht, und ein weiterer Interrupt (sofern aktiviert) kann beim Erreichen von Null gesetzt werden. [Texas Instruments 2018b, S. 339, Kap. 11.2.3.4

[9] Eine Referenz auf ein periodisches Zeitsignal um zeitliche Abläufe zu synchronisieren; typischerweise in Form von Quarzoszillatoren oder externen Taktsignalen.

& S. 361, Kap. 12.2.3.4] Dieser Modus erzeugt eine symmetrische *Pulsweitenmodulation (PWM)*[10] und wird häufig in Anwendungen zur Motorsteuerung oder zur Erzeugung präziser analoger Ausgangssignale eingesetzt. [Texas Instruments 2018b, S. 340, Kap. 11.2.3.5 & S. 362, Kap. 12.2.3.5],[Davies 2008, S. 349, Kap. 8.7]

[Davies 2008, S. 291, Kap. 8.3.1]

Die Wahl eines geeigneten Modus hängt stark von der spezifischen Anwendung ab. Für einfache Zeitmessungen oder periodische Aufgaben ist der Up Mode oft ausreichend, während der Continuous Mode für längere Intervalle oder als Basis für komplexere Zeitsteuerungen dient. Der Up/Down Mode hingegen findet seine Anwendung primär in der Erzeugung von Steuersignalen.

2.1.1.2. Capture-Mode

Der Capture Mode ermöglicht es, den aktuellen Wert des Zählers präzise zu erfassen, wenn ein bestimmtes Ereignis an einem zugehörigen Eingangspin auftritt. Der erfasste Zählerwert wird in einem der Capture-Register (CCR0 bis CCR6) gespeichert. Dies ist besonders nützlich für die Messung von externen Signalen wie Pulsweiten, Frequenzen oder der Zeit zwischen zwei Ereignissen.

Die Timer des MSP430FR5729 unterstützen verschiedene Capture-Modi. Diese legen fest, bei welcher Art von Signaländerung die Erfassung des Zählerwertes erfolgt:

- **Capture on rising edge:** Sobald am zugehörigen Eingangspin eine steigende Flanke detektiert wird (übergang von Low nach High) wird in diesem Modus der aktuelle Zählerwert in das Capture-Register geschrieben.
- **Capture on falling edge:** Hier erfolgt die Erfassung des Zählerwertes am Eingangspin bei einer fallenden Flanke (übergang von High nach Low).
- **Capture on both edges:** Dieser Modus ermöglicht die Erfassung des Zählerwertes sowohl bei steigender als auch fallender Flanken. Dies ist besonders praktisch für die Messung von Signalperioden oder bei Relevanz beider Flanken eines Signals.

[10] Ein Verfahren zur Steuerung der Leistungszufuhr, bei dem die mittlere Ausgangsleistung durch Variieren des Abtastverhältnisses eines Rechtecksignals reguliert wird.

Sofern ein Interrupt im entsprechenden Capture-Register aktiviert ist, kann dieser auch Interrupts auslösen. In der zugehörigen ISR kann der erfasste Zählerwert aus dem Capture-Register gelesen und weiterverarbeitet werden. Mehrere Capture-Register innerhalb eines Timers ermöglichen die Erfassung und Auswertung mehrerer aufeinanderfolgender Ereignisse, ohne dass der vorherige Wert überschrieben wird.

Die Konfiguration des Capture Mode umfasst die Auswahl des auslösenden Ereignisses (Flanke) sowie ggf. die Aktivierung des Capture-Interrupts. Die erfassten Zeitstempel im Capture-Register erlauben präzise Messungen und die Analyse externer Signale in eingebetteten Systemen. [Texas Instruments 2018b, S. 340, Kap. 11.2.4.1 & S. 362, Kap. 12.2.4.1], [Davies 2008, S. 300, Kap. 8.4]

2.1.1.3. Compare-Mode

Der Compare Mode ermöglicht es, den aktuellen Wert des Zählers kontinuierlich mit den in den Compare-Registern CCR0 bis CCR7 hinterlegten Werten zu vergleichen. Wenn der Zählerstand mit dem Vergleichswert übereinstimmt, kann z. B. ein Interrupt ausgelöst oder ein Ausgangspin beeinflusst werden.

Die Compare-Modi bieten verschiedene Möglichkeiten, wie der Ausgangspin bei einer übereinstimmung beeinflusst werden soll:

- **Set output on compare:** Bei einer übereinstimmung des Zählerstandes mit dem Compare-Registerwert wird der zugehörige Ausgangspin auf High gesetzt.
- **Reset output on compare:** Hier wird der Ausgangspin bei übereinstimmung auf Low gesetzt.
- **Toggle output on compare:** In diesem Modus ändert der Ausgangspin bei jeder übereinstimmung seinen Zustand (von High nach Low oder von Low nach High).
- **Output High:** Der Ausgangspin wird permanent auf High gehalten.
- **Output Low:** Der Ausgangspin wird permanent auf Low gehalten.

- **Set/Reset:** In Kombination mit dem Compare-Register 0 (CCR0) kann ein PWM-Signal erzeugt werden. Beispielsweise kann der Ausgang bei Erreichen des CCR0-Wertes gesetzt und bei Erreichen des CCRn-Wertes zurückgesetzt werden (oder umgekehrt), wodurch die Pulsweite durch den Wert in CCRn bestimmt wird.

ähnlich wie beim Capture Mode ermöglicht ein Interrupt der CPU, auf präzise Zeitpunkte zu reagieren und entsprechende Aktionen auszuführen. Der Compare Mode ist somit ein vielseitiges Werkzeug zur Erzeugung von Steuersignalen, zur Implementierung von Zeitverzögerungen oder zur Synchronisation interner Operationen mit einer präzisen Zeitbasis. [Texas Instruments 2018b, S. 342, Kap. 11.2.4.2 & S. 364, Kap. 12.2.4.2], [Davies 2008, S. 352, Kap. 8.8]

Nachdem die verschiedenen Betriebsarten des Timers betrachtet wurden, ist es wichtig zu verstehen, wie die zugehörigen Register konfiguriert werden, um die gewünschte Funktionalität zu erzielen.

2.1.1.4. Einstellungen der Capture and Compare Register

Die Funktionalität der Capture- und Compare-Einheiten wird maßgeblich durch die Konfiguration ihrer zugehörigen Register bestimmt. Hierzu gehören die Aktivierung und Deaktivierung von Interrupts, die Auswahl des Ausgangsmodus (nur für Compare) sowie die Festlegung des auslösenden Ereignisses.

Für jedes Capture/Compare-Register kann individuell festgelegt werden, ob ein Interrupt ausgelöst werden soll, wenn ein entsprechendes Ereignis eintritt. Dies geschieht über spezifische **Interrupt-Enable-Bits** im jeweiligen Capture/Compare-Control-Register (TAxCCTLn oder TBxCCTLn). Durch das Setzen des CCIE-Bits auf Eins oder Null, kann die Generierung eines Interrupts bei einem Capture- oder Compare-Ereignis aktiviert bzw. deaktiviert werden.

Wie bereits im Abschnitt 2.1.1.3 zum Compare Mode beschrieben, legen die Output Mode Bits (OUTMOD) fest, wie der zugehörige Ausgangspin bei einer übereinstimmung des Zählerstandes mit dem Compare-Registerwert beeinflusst wird. Die Auswahl des passenden Output Mode ist entscheidend für die Erzeugung der gewünschten Ausgangssignale, wie beispielsweise bei der Pulsweitenmodulation.

Die Auswahl des auslösenden Ereignisses für eine Capture- oder Compare-Operation wird ebenfalls über Bits im TAxCCTLn- oder TBxCCTLn-Register gesteuert. Für den Capture Mode wird hier beispielsweise mit dem CM-Bit festgelegt, ob die Erfassung bei einer steigenden, fallenden oder beiden Flanken des Eingangssignals erfolgen soll. Im Compare Mode definiert diese Einstellung, unter welchen Bedingungen die Vergleichsoperation als erfolgreich betrachtet wird und die entsprechende Aktion (Interrupt, Ausgangssignaländerung) ausgelöst wird. Dies kann beispielsweise ein reiner Vergleich oder auch ein Vergleich in Kombination mit dem überlauf des Zählers im Up Mode sein. [Texas Instruments 2018b, S. 351, Kap. 11.3.3 & S. 375, Kap. 12.3.3], [Davies 2008, S. 292, Kap. 8.3.2]

Die sorgfältige Konfiguration dieser Einstellungen in den Capture/Compare-Registern ist unerlässlich, um den Timer präzise an die Anforderungen der jeweiligen Applikation anzupassen.

Ein weiterer fundamentaler Aspekt der Timer-Konfiguration ist u. a. die Wahl der Taktquelle, welche die Zeitbasis für den Zähler und somit für alle zeitgesteuerten Operationen des Timers bestimmt.

2.1.1.5. Timer Control-Register

Die Timer des MSP430FR5729 können von verschiedenen internen Taktquellen getaktet werden, die jeweils unterschiedliche Eigenschaften und Anwendungsbereiche aufweisen. Die primären Taktquellen sind *Auxiliary Clock (ACLK)*[11] und *Sub-Main Clock (SMCLK)*[12]. Auch externe Taktquellen können zur Taktung des Timers herangezogen werden wie z. B. das TACLK/TBCLK-Register oder der INCLK-Pin. [Texas Instruments 2018b, S. 71, Kap. 3.1] [Davies 2008, S. 163, Kap. 5.8 & S. 289, Kap. 8.3.1]

Die Auswahl der Clock-Source für einen Timer erfolgt über spezifische Bits im TAxCTL oder TBxCTL Timer Control Register. Das TASSEL-/TBSSEL-Bit legt fest, ob der Timer von TAxCLK/TBxCLK, ACLK, SMCLK oder INCLK getaktet wird.

[11] Niederfrequente Taktquelle in Mikrocontroller-Systemen, die typischerweise von einem Quarzoszillator gespeist wird und für energiesparende Betriebsmodi verwendet wird.
[12] Taktgesteuertes Signal, das typischerweise für Peripheriegeräte verwendet wird und sich aus einer frei wählbaren Taktquelle ableiten lässt.

Die Wahl der Clock-Source hat einen direkten Einfluss auf die Timer-Frequenz, wobei die Timer-Frequenz nicht gleich der Frequenz der gewählten Clock-Source entsprechen muss. Durch optionale Prescaler-Werte wie dem ID-Bit und dem TAIDEX-/TBIDEX-Bit kann die Frequenz weiter individualisiert werden. [Texas Instruments 2018b, S. 349, Kap. 11.3.1 & S. 372, Kap. 12.3.1] [Davies 2008, S. 289, Kap. 8.3.1]

Die Timer-Frequenz bestimmt wiederum die Zeitbasis des Timers. Eine höhere Timer-Frequenz führt zu einer feineren Zeitauflösung, da der Zähler schneller inkrementiert wird. Dies ermöglicht präzisere Zeitmessungen und die Erzeugung von Signalen mit höherer Frequenz. Umgekehrt führt eine niedrigere Frequenz zu einer gröberen Zeitauflösung, kann aber den Stromverbrauch reduzieren.

Ein weiteres Steuerbits wie das **Mode Control-Bit (MC)** steuert die bereits in Kapitel 2.1.1.1 erläuterten Zähl-Modi und das TAIE-/TBIE-Bit steuert, ob Interrupts Ein- oder Ausgeschaltet sind.

Die Auswahl der Clock-Source, des Prescalers und weiteren Steuerbits ist daher entscheidend, um die gewünschte Zeitbasis, Auflösung und Verhalten für den zu konfigurierenden Timer zu erreichen um die Anforderungen der jeweiligen Anwendung optimal zu erfüllen.

2.1.2. Fazit und Einsatzmöglichkeiten

Die detaillierte Auseinandersetzung mit der Timer-Architektur des MSP430FR5729 hat die Flexibilität und Leistungsfähigkeit dieser Peripheriekomponente verdeutlicht. Die Unterscheidung zwischen Timer des Typs A und B, die verschiedenen Betriebsarten (Count, Capture, Compare) sowie die vielfältigen Einstellmöglichkeiten der Capture/Compare-Register und die Auswahl der Taktquelle eröffnen ein breites Spektrum an Anwendungsmöglichkeiten in eingebetteten Systemen.

Analog zur Übersicht "What Timer Where?"von John H. Davies lassen sich die primären Einsatzgebiete der Timer des MSP430FR5729 wie folgt zusammenfassen:

- **Zeitmessung und Zeitbasis:** Unabhängig vom Timer-Typ können alle als eine präzise Zeitbasis dienen. Durch die Wahl einer geeigneten Clock-Source und eines passenden Prescalers lassen sich genaue Zeitintervalle festlegen. Dies ist fundamental für das Zeitmanagement innerhalb des Mikrocontrollers und die Synchronisation mit externen als auch Internen Ereignissen. Timer A eignet

sich hierbei oft für grundlegende Zeitsteuerungsaufgaben, während die flexiblere Konfigurierbarkeit des Timers vom Typ B wie z. B. verschiedene Bit-Längen (2.1.1) eine feinere Anpassung an spezifische Zeitmessanforderungen erlaubt.

- **Ereigniserfassung (Capture):** Die Capture-Funktionalität ermöglicht die präzise Erfassung des Zeitpunkts externer Ereignisse. Dies ist unerlässlich für Anwendungen wie die Messung von Pulsweiten, die Frequenzmessung von Signalen oder die Erfassung der Ankunftszeit von Informationen in Kommunikationsprotokollen. Die Möglichkeit, sowohl steigende, fallende Flanken oder auch beide zu erfassen, erweitert den Anwendungsbereich in verschiedenen Szenarien deutlich.

- **Signalerzeugung (Compare/PWM):** Die Compare-Einheiten in Verbindung mit den verschiedenen Ausgangsmodi erlauben die Generierung präziser Ausgangssignale. Dies ist besonders relevant für die Pulsweitenmodulation, die zur Steuerung von Motoren, zur Dimmung von LEDs oder zur Erzeugung analog wirkender Signale eingesetzt wird. Der Up/Down Mode des Count-Modus in Kombination mit den Compare-Registern des Timer B bietet hierbei besonders flexible Möglichkeiten zur Erzeugung verschiedenster PWM-Signale.

- **Interrupt-Steuerung:** Sowohl Capture- als auch Compare-Ereignisse können Interrupts auslösen. Dies ermöglicht eine effiziente Reaktion des Mikrocontrollers auf zeitgesteuerte Ereignisse oder externe Signale, ohne die kontinuierliche abfrage des Timer-Status. Die präzise Interrupt-Generierung trägt maßgeblich zur Realisierung reaktiver und effizienter eingebetteter (Echtzeit-) Systeme bei.

Zusammenfassend lässt sich der grundlegende Aufbau eines Timers, vereinfacht nach dem Vorbild von Abbildung 8.5 aus Davies' Buch, wie folgt darstellen:

Ein Timer besteht im Kern aus einem Zähler (2.1.1.1), der durch eine ausgewählte Clock-Source (2.1.1.5) in definierten Schritten inkrementiert oder dekrementiert wird. Dieser Zähler läuft gemäß der gewählten Betriebsart.

Zusätzlich verfügt der Timer über Sieben Capture/Compare-Kanäle. Jeder Kanal beinhaltet mindestens ein Capture/Compare-Register und eine zugehörige Steuereinheit.

Im Capture Mode (2.1.1.2) wird der aktuelle Wert des Zählers in das CCRx-Register geschrieben, wenn ein durch die Steuereinheit ausgewähltes Ereignis (z. B. Flanke an einem Eingangspin) eintritt.

Im Compare Mode (2.1.1.3) wird der aktuelle Wert des Zählers kontinuierlich mit dem Wert im CCRx-Register verglichen. Bei einer Übereinstimmung löst die Steuereinheit eine konfigurierte Aktion aus, wie beispielsweise das Setzen/Rücksetzen/Toggeln eines zugehörigen Ausgangspins oder die Generierung eines Interrupts, sofern dieser in der Steuereinheit aktiviert wurde.

Die Steuereinheit ermöglicht die Konfiguration des jeweiligen Kanals, einschließlich der Auswahl des Capture/Compare-Modus, des auslösenden Ereignisses, des Ausgangsmodus und der Aktivierung/Deaktivierung des Interrupts.

Die Darstellung Abbildung 2.2 des Timer B als Block Diagram verbildlicht die grundlegenden Komponenten eines Timer-Kanals und deren Zusammenspiel. Die flexiblen Konfigurationsmöglichkeiten dieser einzelnen Blöcke ermöglichen die Realisierung einer Vielzahl von Zeitsteuerungs- und Signalverarbeitungsaufgaben in eingebetteten Systemen mit dem MSP430FR5729.

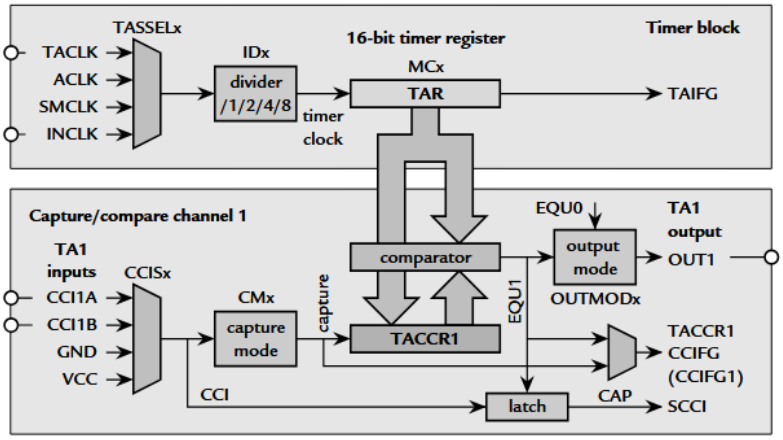

Abb. 2.2.: Timer B Block & Capture/Compare Channel 1 [Davies 2008, S. 355, Kap. 8.9]

2.1.3. Enhanced Universal Serial Communication Interface (eUSCI)

2.2. Konzeprionierung Observer-Modul

2.3. Interruptgesteuertes Lesen und Schreiben

2.4. Debugging-Methoden: Hardware- vs. Software-Breakpoints

Im Kontext der Fehlersuche und Programmanalyse in Embedded Systems stellen *Breakpoints*[13] ein fundamentales Werkzeug dar. Sie ermöglichen es, die Ausführung eines Programms an einer vordefinierten Stelle zu unterbrechen, um den internen Zustand des Systems zu inspizieren. Grundsätzlich lassen sich zwei primäre Arten von Breakpoints unterscheiden: Software-Breakpoints und Hardware-Breakpoints, deren Implementierung und Eigenschaften sich signifikant unterscheiden.

Software-Breakpoints werden, wie der Name impliziert, zur aktiven Laufzeit des Programms durch einen direkten Eingriff in den ausführbaren Code im Speicher des Mikrocontrollers realisiert. An der Zieladresse wird hierbei die ursprüngliche Programminstruktion temporär durch eine spezielle Instruktion ersetzt – oftmals ein definierter Opcode, der einen Software-Interrupt oder eine Exception auslöst (z. B. eine Breakpoint-Instruktion oder ein anderer reservierter Trap-Befehl). Sobald der Programmzähler diese modifizierte Stelle erreicht und die spezielle Instruktion ausgeführt wird, unterbricht der Mikrocontroller den normalen Programmfluss und springt in eine Debug-Routine. Der *Debugger*[14] kann diesen Zustand erkennen, die ursprüngliche Instruktion wiederherstellen und dem Entwickler die Kontrolle übergeben. Durch diesen Mechanismus sind Software-Breakpoints hochflexibel und können

[13] Bezeichnet in der Softwareentwicklung eine vom Entwickler bewusst gesetzte Unterbrechung im Programmablauf, die typischerweise zur Laufzeit-Debugging-Zwecken verwendet wird. Beim Erreichen dieses Punkts wird die Ausführung des Programms angehalten, sodass der aktuelle Zustand (z. B. Variableninhalte, Stack, Speicher) analysiert werden kann.

[14] Ein Werkzeug zur schrittweisen Ausführung und Analyse von Programmen. Es erlaubt das Setzen von Haltepunkten, das überprüfen von Speicherinhalten und das Nachvollziehen von Kontrollflüssen zur Fehlersuche und -behebung.

an nahezu jeder beliebigen Stelle im beschreibbaren Code-Speicher (wie RAM oder, im Falle des MSP430FR5729, auch im FRAM) gesetzt werden.

Die Vorteile des softwarebasierten Ansatzes liegen primär in der Möglichkeit, eine praktisch unbegrenzte Anzahl von Breakpoints im System zu verwenden, sowie in den geringen Anforderungen an zusätzliche, dedizierte Hardwarekomponenten auf dem Zielsystem selbst. Die grundlegende Fähigkeit der Zentralen Recheneinheit, Interrupts oder Exceptions zu behandeln, ist hierfür ausreichend.

Gegenüber den Software-Breakpoints bieten hardwarebasierte Breakpoints den Vorteil, dass sie Programmunterbrechungen auch in solchen Speichersegmenten ermöglichen, die schreibgeschützt sind (z.B. ROM oder spezifisch geschützte Flash-Bereiche). Ein weiterer entscheidender Vorteil ist ihre Nicht-Intrusivität: Da keine Modifikation des Programmcodes stattfindet, werden weder die Konsistenz des Codes im Speicher noch das präzise Echtzeitverhalten (Timing) des Programms durch den Breakpoint-Mechanismus selbst beeinflusst.

Um dies jedoch zu erreichen, benötigen Hardware-Breakpoints ein dediziertes Hardwaremodul innerhalb des Mikrocontrollers. Im Falle der MSP430-Mikrocontrollerfamilie ist dies das **Embedded Emulation Module (EEM)** [Texas Instruments 2018b, S. 569, Kap. 21]. Dieses Modul beinhaltet spezielle Hardwareregister, typischerweise Adresskomparatoren, welche die Speicheradresse des Befehls halten, an welcher bei übereinstimmung mit dem Programmzähler ein Breakpoint ausgelöst werden soll. Da der Breakpoint durch externe Hardwarelogik ausgelöst wird und nicht durch eine im Programmablauf ausgeführte Instruktion, müssen seitens des Breakpoint-Mechanismus selbst keine Registerinhalte oder Stack-Elemente explizit zwischengespeichert und im Nachhinein wiederhergestellt werden. Ganz im Unterschied wie es bei einem durch einen Software-Breakpoint induzierten Interrupt der Fall sein kann. Die Zustandsicherung erfolgt erst durch die Debug-Routine nach erfolgter Unterbrechung. Der wesentliche Nachteil hierbei ist allerdings die strikte Limitierung der Anzahl gleichzeitig setzbarer Hardware-Breakpoints, welche direkt von der Anzahl der im EEM verfügbaren Komparator-Register abhängt. Für den MSP430 sind dies oft nur zwei oder drei [Texas Instruments 2025, vgl. Kap. 7.1].

Diese Gegenüberstellung offenbart einen klaren Trade-off: Hardware-Breakpoints glänzen durch ihre Transparenz und die Fähigkeit, in geschützten Speicherbereichen zu operieren, sind jedoch eine knappe Ressource. Software-Breakpoints hingegen bieten eine hohe Flexibilität und nahezu unbegrenzte Verfügbarkeit, gehen

aber mit einer leichten Modifikation des Programmcodes und potenziellen, wenn auch meist minimalen, Timing-Veränderungen einher. Angesichts der begrenzten Anzahl an Hardware-Breakpoints auf der MSP430-Plattform, die insbesondere bei komplexeren Debugging-Szenarien schnell erschöpft sein können, erweist sich die Implementierung von Software-Breakpoints als eine pragmatische und oft notwendige Erweiterung der Debugging-Möglichkeiten. Um die technischen Rahmenbedingungen für die Realisierung solcher Software-Breakpoints auf dem MSP430FR5729 sowie die Interaktion mit der Debugging-Infrastruktur genauer zu verstehen, ist eine detaillierte Betrachtung des eingesetzten Debug-Adapters und seiner Funktionsweise unerlässlich. Die nachfolgende Analyse des MSP-FET Debuggers wird diese Aspekte beleuchten und die Grundlage für die spätere Implementierungsstrategie legen.

2.4.1. Der MSP-FET Debugger im Detail

Die effektive Nutzung von sowohl Hardware- als auch Software-Breakpoints auf dem MSP430FR5729 ist maßgeblich von der externen Debugging-Hardware und -Software abhängig. Als zentrale Schnittstelle zwischen der Entwicklungsumgebung auf dem Host-PC und dem Ziel-Mikrocontroller dient in diesem ökosystem der **MSP-FET (Flash Emulation Tool) Debugger**. Dieses externe Gerät stellt die physische und logische Verbindung zum MSP430 her und ermöglicht tiefgreifende Eingriffe und Beobachtungen während der Programmausführung.

Der MSP-FET kommuniziert mit dem MSP430-Mikrocontroller typischerweise über standardisierte (Debug-)Schnittstellen wie **JTAG (Joint Test Action Group)** oder das von Texas Instruments entwickelte *Spy-Bi-Wire (SBW)*[15]. über diese Schnittstellen erhält der MSP-FET Zugriff auf das EEM des MSP430FR5729. Wie im vorherigen Abschnitt 2.4 dargelegt, ist das EEM für die Realisierung von Hardware-Breakpoints zuständig. Der MSP-FET agiert hierbei als Vermittler, der die vom Entwickler in der **IDE (Integrated Development Environment)** gesetzten Hardware-Breakpoint-Adressen in die entsprechenden Register des EEM schreibt und die vom EEM generierten Haltesignale empfängt und an die IDE weiterleitet. Somit ist der MSP-FET unerlässlich für die Konfiguration und Nutzung der limitierten, aber präzisen Hardware-Breakpoint-Ressourcen des Mikrocontrollers.

[15] Zweidraht-Variante des JTAG-Protokolls, die Pin-Anzahl am Target reduziert und besonders für platzkritische Anwendungen von Vorteil ist.

Darüber hinaus spielt der MSP-FET eine ebenso große Rolle für die Implementierung und Handhabung von Software-Breakpoints. Die Fähigkeit, den Speicher des MSP430FR5729 (sowohl RAM als auch das beschreibbare FRAM) zur Laufzeit zu lesen und zu schreiben, ist die Grundvoraussetzung, um Instruktionen mit einer Software-Breakpoint-Routine zu ersetzen. Der Debugger liest über den MSP-FET die ursprüngliche Instruktion an der Zieladresse aus, ersetzt diese durch eine Breakpoint-Instruktion und, nach dem Auslösen des Software-Interrupts, stellt er die ursprüngliche Instruktion wieder her. Ferner ermöglicht der MSP-FET die Steuerung des Programmflusses (Anhalten, Starten, Einzelschrittbetrieb) und den Zugriff auf CPU-Register und Speicherinhalte, was für die Analyse des Systemzustands an einem Breakpoint unerlässlich ist. Das vom Software-Breakpoint ausgelöste Interrupt- oder Exception-Signal wird ebenfalls über die Debug-Schnittstelle an den MSP-FET und somit an die Host-Debugger-Software gemeldet.

Es ist wichtig zu verstehen, dass der MSP-FET primär die Kommunikationsinfrastruktur und die Low-Level-Zugriffsmechanismen bereitstellt. Während er das Setzen von Hardware-Breakpoints direkt über das EEM steuert, stellt er für Software-Breakpoints die notwendigen Lese-, Schreib- und Kontrolloperationen zur Verfügung. Die eigentliche Logik eines Software-Breakpoints – das heißt, welche Instruktion als Breakpoint-Befehl dient, wie die ursprüngliche Instruktion gesichert und wiederhergestellt wird sowie der resultierende Trap behandelt wird – muss in der Debugger-Software auf dem Host und gegebenenfalls durch eine minimale Debug-Monitor-Routine auf dem Target implementiert werden, wobei der MSP-FET als Brücke dient.

Die Kenntnis der Funktionalitäten und der Arbeitsweise des MSP-FET ist somit entscheidend für die Entwicklung einer robusten Software-Breakpoint-Lösung. Er definiert die Grenzen und Möglichkeiten, wie mit dem Target-System interagiert werden kann, um Breakpoints zu setzen, Zustandsinformationen abzufragen und die Programmausführung zu steuern. Die im Folgenden zu entwickelnde Strategie zur Implementierung von Software-Breakpoints muss sich daher eng an den durch den MSP-FET und die Debug-Schnittstelle des MSP430FR5729 gegebenen Rahmenbedingungen orientieren.

2.4.2. Implementierung von Software-Breakpoints

Zur Realisierung von Breakpoints existieren mehrere Ansätze. Ein bewährter Einstieg besteht darin, etablierte Debugger und ihre Architektur zu studieren. Im Embedded- und Low-Power-Bereich kommen beispielsweise Werkzeuge wie **TRACE32**, **M-Core** oder das **MSP-FET** von Texas Instruments zum Einsatz. Diese Debugger setzen Hardware-Breakpoints über spezielle Debug-Interfaces um und bieten damit eine hohe Zuverlässigkeit bei minimaler Eingriffstiefe in das Laufzeitsystem.

Im Gegensatz dazu zielt die hier vorgestellte Lösung auf Software-Breakpoints ab, die direkt den Programmspeicher manipuliert. Dabei wird, in dieser Implementierung, an der gewünschten Halteadresse der originale Maschinenbefehl durch einen Sprungbefehl (Jump) ersetzt, der auf eine speziell implementierte **Breakpoint-Handler**-Routine verweist. Beim Erreichen dieses Befehls wird zunächst ein kritischer Abschnitt eingeleitet: Es werden die für den Prozess relevanten Register – namentlich *Program Counter (PC)*[16], *Stack Pointer (SP)*[17], *Statusregister (SR)*[18] und ggf. mehrere **General-Purpose-Register (R4 bis R15)** – gesichert und Interrupts deaktiviert, um eine atomare Kontextsicherung zu gewährleisten [Texas Instruments 2018b, S.91, Kap. 4.3]. Anschließend erfolgt der übergang in die Handler-Routine, die das gesamte System bis auf das Observer-Modul blockiert und so das Auslesen und Manipulieren von Speicherinhalten ermöglicht.

Nach der Analyse kann der ursprüngliche Programmzustand durch Wiederherstellen der Registersätze wiederhergestellt werden. Eine explizite aktivierung der Interrupts ist daher nicht nötig. Der Compiler stellt hierfür *Compiler-Intrinsics*[19] bereit wie **___get_SR()**, **___get_SP()** und **___set_interrupt_state()** [Texas Instruments 2018a, S.137, Kap. 6.8.1]. Auf diese Weise wird ein vollständiger Zyklus von Unterbrechung, Inspektion und Fortsetzung des Programmflusses realisiert, ohne dass das Hauptprogramm von der Existenz des Observer-Moduls Kenntnis erhält.

Vor diesem Hintergrund wird im folgenden Abschnitt die Konzeptionierung einfacher Software-Breakpoints im Detail erläutert und auf die dafür notwendigen Voraussetzungen eingegangen.

[16] Ein Register, das die Speicheradresse des derzeitigen Befehls enthält.
[17] Ein Register, das die Speicheradresse des letzten oder ersten Datenelements im Stack speichert.
[18] Register für eine Reihe von Flags, die von der arithmetisch-logischen Einheit in Abhängigkeit der zuletzt durchgeführten Rechenoperation gesetzt werden.
[19] Compiler-spezifische, vordefinierte Funktionen, die direkt in optimierten Assemblercode umgesetzt werden.

2.4.3. Konzeptionierung Simpler Software Breakpoints

Die Grundidee einfacher Software-Breakpoints besteht darin, an einer Stelle im Programmspeicher, an der ein gültiger Maschinenbefehl *(Opcode)*[20] liegt, diesen Befehl temporär durch einen Sprung auf die Breakpoint-Handler-Routine zu ersetzen. Zunächst wird der originale Opcode gesichert, um ihn später unverändert wieder einsetzen zu können. Die Auswahl der Adresse erfordert, dass diese auf ein gültiges Befehlswort ausgerichtet ist und im Stack-Bereich liegt, um Kollisionen auf den Code-Segmenten zu vermeiden.

Die Implementierung gliedert sich in folgende Schritte:

1. **Adressvalidierung:** Prüfen, ob die Zieladresse auf ein Doppelwort (8 Bytes) ausgerichtet ist und im RAM/FRAM-Stack liegt.
2. **Kontext-Sicherung:** In einem kritischen Abschnitt werden PC, SP, SR und alle modifizierenden Register in einem Puffer abgelegt.
3. **Ersetzen des Opcodes:** Der Original-Opcode (8 Bytes) wird durch den 8-Byte-Sprungbefehl (4 Bytes für den Instruction-Code, 4 Bytes für die Zieladresse der Handler-Routine) überschrieben [Texas Instruments 2018b, S.161, Kap. 4.6.2.28].
4. **Handler-Ausführung:** Beim Eintreten der Breakpoint-Adresse springt der PC in die Handler-Routine, die das System an der definierten Stelle anhält und das Observing-Modul für weitere Debug-Schritte aktiv hält.
5. **Kontext-Wiederherstellung:** Nach Abschluss der Debug-Aktion werden alle Register und der ursprüngliche Opcode wiederhergestellt, bevor der normale Programmablauf fortgesetzt wird.

Diese modulare Aufteilung stellt die Rückkehr zum ursprünglichen Systemzustand – einschließlich des Originalen-Opcodes – sicher und garantiert die Konsistenz des Programms.

Um die im vorigen Abschnitt 2.4.2 skizzierten Konzepte robust umzusetzen, sind im nächsten Abschnitt technische Details wie Instruktionslängen und Speicher-Alignment zu betrachten.

[20] Auch op code oder operation code, ist eine meist in hexadezimaler Schreibweise angegebene Zahl, die die Nummer eines Maschinenbefehls für einen bestimmten Prozessortyp angibt.

2.4.4. Instruktionslängen und Speicher-Alignment

Die Manipulation von Befehlen im Stack ist hoch kritisch, da das Hauptprogramm keine Kenntnis vom Observer-Modul hat und eine falsche Adressierung oder unvollständige Opcode-substituierung zu undefiniertem Verhalten führen kann. Zwei zentrale Aspekte sind dabei zu beachten:

- **Speicher-Alignment:** MSP430-Instruktionen sind an geraden Adressen ausgerichtet. Vor jedem Schreibzugriff muss daher sichergestellt werden, dass die Zieladresse gerade ist. Ungerade Adressen führen zu einer Fehlerausgabe.
- **Opcode-Längen:** Während einzelne Maschinenbefehle in der Regel 4 Bytes belegen, können komplexe Instruktionen – etwa `MOV.W` – bis zu 12 Bytes lang sein [Texas Instruments 2018b, S.165, Kap. 4.6.2.32]. Diese Variabilität erschwert den Austausch der 8 Bytes (Jump-Instruktion plus Zieladresse), da unter diesen Randbedingungen leicht zu viele oder zu wenige Bytes überschrieben werden und so ungültige oder ungewollte Instruktionen entstehen.

Das Sichern und Wiederherstellen von Registern und des Originalen Opcodes reicht daher nicht aus. Es wird eine Kopie des Stacks benötigt, in welcher der bestehende Opcode Manipuliert wird. Dies garantiert ein sicheres zurückkehren in die Hauptroutinen, sowie eine Robuste Ausführung der Funktion zum verarbeiten der ggf. mehreren Breakpoints.

Dies erschwert die Umsetzung und erhöht die Komplexität der Routine erheblich, wodurch Timing und Konsistenz gefährdet werden. Im anschließenden Fazit werden die gewonnenen Erkenntnisse bewertet, offene Fragestellungen skizziert und ein Ausblick auf weiterführende Arbeiten gegeben.

2.4.5. Fazit zur Umsetzung von Software Breakpoints

Die Realisierung von Software-Breakpoints auf einem Low-Power-Mikrocontroller wie dem MSP430FR5729 erfordert ein tiefgehendes Verständnis der Prozessor-Architektur, der Instruktionsformate und der nebenläufigen Abläufe im System. Es müssen kritische Bereiche atomar bearbeitet, Register und Stack-Zustände zuverlässig gesichert und Intrinsics korrekt eingesetzt werden. Zudem sind umfangreiche Subsysteme zur überwachung und Protokollierung des Systemzustands zu implementieren.

Diese Komplexität, gepaart mit den Anforderungen an Robustheit, Echtzeitfähigkeit und geringst möglichen Eingriff in den Betrieb, überschreitet den Rahmen einer Bachelorarbeit deutlich. Eine vollständige, produktionsreife Implementierung würde den Umfang einer Masterarbeit oder vergleichbarer Forschungsarbeiten annehmen, in denen Zeit für detaillierte Prototypen, umfangreiche Tests und Performance-Optimierungen eingeplant werden kann. Dennoch bildet dieses Thema eine exzellente Grundlage für weiterführende Arbeiten in den Bereichen eingebettete Echtzeitsysteme und Debugging-Technologien.

3. Fazit und kritische Bewertung

3.1. Das Ergebnis

Der Relaunch des gesamten Webauftritts von HIT RADIO FFH erfolgte am 24. Juli 2011, mit dem auch der neue Webradio-Player zum ersten Produktiveinsatz gelangte. In Abbildung 3.1 ist die Webradio-Player-Extension im Live-Einsatz auf webradio.ffh.de zu sehen.

Abb. 3.1.: Der FFH Webradio-Player im Live-Einsatz

Die grundsätzliche Entscheidung, bei der Extension-Entwicklung auf die Frameworks Extbase und Fluid zu setzen, hat sich als absolut richtig erwiesen. Trotz der vielen Kinderkrankheiten und der unvollständigen Implementation vieler Funktionen ist die anwendungsdomänen-getriebene Herangehensweise von Extbase bzw. FLOW3 ein deutlicher Schritt in eine einfachere Richtung der Entwicklung.

Der Webradio-Player läuft seit dem Relaunch nahezu fehlerfrei und benötigte danach nur kleine Anpassungen wegen Schnittstellenänderungen des Stream-Providers Nacamar.

3.2. Die Bewertung der Frameworks Extbase und Fluid

Das bei der Radio/Tele FFH GmbH im Zuge des Relaunches zum Einsatz kommende Content Management System TYPO3 bildete die Basis der Extension-Entwicklung während dieser Arbeit. Aufgrund der angepriesenen Zukunftssicherheit wurde zudem entschieden, alle Extensions mit denen am Markt neu eingeführten Frameworks Extbase und Fluid zu entwickeln. Sowohl Extbase als MVC und Domain-Driven Design basiertes PHP-Framework, als auch Fluid als XML basierte Templating-Engine vereinen das gemeinsame Prinzip »Convention over Configuration«. Dieses Prinzip stellt viele Entwickler vor die große Herausforderung, ihre bisherigen Gewohnheiten beim Entwickeln von Software zu überdenken.

Die vielen zunächst übertrieben erscheinenden Konventionen beim Programmieren mit Extbase und Fluid haben schlussendlich trotz vieler Zweifel ermöglicht, dass drei Entwickler parallel an verschiedenen Extensions arbeiten konnten, auch wenn sie nicht am ursprünglichen Planungsprozess beteiligt waren. Einzig durch die strengen Konventionen in der Programmierung gelingt es, sich ohne große Probleme in andere Extensions einzulesen. So war es beispielsweise möglich, bestimmte Quellcode-Fragmente schnell in andere Extensions zu portieren, ohne viel am bereits vorhandenen Code ändern zu müssen.

Die Templating-Engine Fluid übernimmt diese hervorragenden Eigenschaften bei der Umsetzung in der View-Ebene der Extensions. Als XML-basierte Templating-Engine ist sie flexibel einsetzbar, erzeugt validen XHTML-Code und ist durch ihre Viewhelper theoretisch unendlich erweiterbar. Die bereits implementierten Viewhelper zeugen von der Macht, die von dieser Engine ausgeht. Aus diesem Grund ist es schade, dass bisher nur Basisfunktionen in Viewhelpern vorliegen. Geht es darum, exotischere Logiken in Viewhelpern zu verwenden, so ist man leider darauf angewiesen, diese selbst zu programmieren. Hierdurch geht ein nicht unerheblicher Teil an Entwicklungszeit verloren, weil die Rahmenbedingungen für die Umsetzung TYPO3 interner Funktionen leider meistens schlecht sind. Es fehlt an Dokumentationen der TYPO3 Funktionalitäten, aber auch an Möglichkeiten, diese in Viewhelpern zu realisieren. Aus diesem Grund ist man auch hier stark auf die Community im Internet angewiesen. Den Hauptanteil an Informationen bezieht man von anderen Ent-

wicklern, die sich die Lösungen meist umständlich durch *trial & error*[21] erarbeitet haben.

Theoretisch lässt sich Fluid mit jedem verfügbaren PHP-Framework verwenden. So kann eine simple Webseite auch komplett autark ohne Content Management System aufgebaut werden.

Zusammenfassend bleibt zu sagen, dass sowohl Extbase als auch Fluid ihren Weg in die Welt von TYPO3 gefunden haben. Die beiden alternativen Frameworks geben einen Vorgeschmack auf die Entwicklung mit TYPO3 v5 und FLOW3 und helfen Entwicklern schon heute sich auf die Portierung ihrer Extensions vorzubereiten. Für den Autor als Neueinsteiger in der Extension-Entwicklung boten die beiden Werkzeuge viele neue und intuitive Wege, an die Entwicklung von Software heranzugehen. Nicht zuletzt ist hervorzuheben, dass die neuartigen Ansätze der Entwicklung, wie etwa das Domain-Driven Design und Convention over Configuration, viel Erleichterung in den Alltag eines Software-Entwicklers bringen können, sofern dieser sich darauf einlässt.

3.3. Ein Ausblick

Es hat sich gezeigt, dass die Entwicklung der Frameworks Extbase und Fluid ständig voranschreitet, weshalb es in unregelmäßigen Abständen sinnvoll ist, die Webradio-Player-Extension zu aktualisieren und Neuerungen direkt zu implementieren. Dadurch ist eine spätere Umsetzung in FLOW3 für TYPO3 Version 5 einfacher zu realisieren.

Durch die grundverschiedenen Ansätze beider Varianten ist schnell erkennbar, dass mit der Entwicklung Schritt gehalten werden muss, damit die eigenen Extensions zukunftssicher bleiben. Aus diesem Grund werden alle bei der Radio/Tele FFH GmbH entwickelten Extensions – so auch der Webradio-Player – ständig auf dem neusten Entwicklungsstand gehalten.

Einer Portierung auf FLOW3 stünde somit nichts im Wege.

[21] Heuristische Methode, bei der durch Versuch und Irrtum eine Lösung gefunden wird.

Literaturverzeichnis

Davies 2008

Davies, John H.: *MSP430 Microcontroller Basics*. 1. Oxford : Newnes, 2008. – ISBN 978–0–7506–8276–3 2.1.1, 2.1.1.1, 2.1.1.2, 2.1.1.3, 2.1.1.4, 2.1.1.5, 2.2, 3.3

Texas Instruments 2017

Texas Instruments: *MSP430FR5729 Mixed-Signal Microcontroller*. Revision C. 655303 Dallas, Texas: Texas Instruments Incorporated, 2017. – 119 S. https://www.ti.com/lit/ds/symlink/msp430fr5729.pdf. – Document Number: SLASE35 2.1, 2.1, 3.3

Texas Instruments 2018a

Texas Instruments: *MSP430 Optimizing C/C++ Compiler v18.1.0.LTS*. Revision R. 655303 Dallas, Texas: Texas Instruments Incorporated, 2018. – 181 S. https://www.ti.com/lit/ug/slau132r/slau132r.pdf. – Document Number: SLAU132, Rev. R 2.4.2

Texas Instruments 2018b

Texas Instruments: *MSP430FR57xx Family User's Guide*. Revision D. 655303 Dallas, Texas: Texas Instruments Incorporated, 2018. – 576 S. https://www.ti.com/lit/ug/slau272d/slau272d.pdf. – Document Number: SLAU272, Rev. D 2.1.1, 2.1.1.1, 2.1.1.2, 2.1.1.3, 2.1.1.4, 2.1.1.5, 2.4, 2.4.2, 3, 2.4.4

Texas Instruments 2025

Texas Instruments: *Code Composer Studio™ User's Guide*. Version 20.1.1. 655303 Dallas, Texas: Texas Instruments Incorporated, 2025. https://software-dl.ti.com/ccs/esd/documents/users_guide/index.html. – Document Number: SLAU132, Rev. R 2.4

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Eidesstattliche Versicherung

Ich, Julian Rapp, Matrikel-Nr. 304875, versichere hiermit, dass ich meine Bachelor-Thesis mit dem Thema

> *Implementierung einer interruptgesteuerten Benutzerschnittstelle auf einem Low-Power-Mikrocontroller*

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Mir ist bekannt, dass ich meine Bachelor-Thesis zusammen mit dieser Erklärung fristgemäß nach Vergabe des Themas in dreifacher Ausfertigung und gebunden im Prüfungsamt der Hochschule für Technik, Wirtschaft und Gestaltung abzugeben oder spätestens mit dem Poststempel des Tages, an dem die Frist abläuft, zu senden habe.

Konstanz, den 10. Mai 2025

Julian Rapp

A. Anhang

A.1. Beiliegende CD

A.1.1. Inhaltsverzeichnis der CD

1. Die gesamte Bachelorarbeit als PDF-Datei
2. Alle verwendeten Online-Quellen als PDF-Ausdruck
3. Sonstige Quelltexte
4. Alle in der Bachelorarbeit verarbeiteten Diagramme
5. Alle in der Bachelorarbeit verarbeiteten Grafiken
6. Alle in der Bachelorarbeit verarbeiteten Screenshots