# Was wurde erreicht?

In diesem selbstständig durchgeführten Projekt habe ich erfolgreich einen Mikroprozessor in der Programmiersprache C programmiert, um eine Stoppuhr und einen Timer zu realisieren, die gleichzeitig laufen können. Das Hauptziel bestand darin, die Fähigkeiten des Mikroprozessors zu nutzen und meine Programmierkenntnisse in C anzuwenden, um eine zuverlässige Zustandsmaschine zu implementieren.

Ich begann das Projekt mit einer gründlichen Planung, bei der ich Klassendiagramme, State-Event-Diagramme und Komponentendiagramme erstellte. Diese Diagramme halfen mir, den strukturierten Aufbau des Projekts zu visualisieren und eine klare Richtlinie für die Implementierung zu schaffen. Die Diagramme wurden mit PlantUML erstellt und während der Implementierung mehrmals überarbeitet, um die Anforderungen zu erfüllen und die Funktionalität zu verbessern.

Um eine bessere Organisation und Modularität zu gewährleisten, habe ich die Aufgaben in mehrere Tasks und Dateien unterteilt. Dadurch konnte ich die Dateien problemlos als Bibliotheken in andere Projekte übertragen und wiederverwenden. Diese modulare Struktur erleichterte auch meine eigene Arbeit und ermöglichte es mir, verschiedene Aufgaben parallel zu bearbeiten.

Durch das Projekt habe ich wertvolle Kenntnisse über Tasks, Semaphoren und Multitasking erworben. Durch die Verwendung von Prioritäten konnte ich die Ausführung von Tasks steuern und eine effiziente Ressourcennutzung sicherstellen. Mit den Funktionen "osDelay" und "osDelayUntil" konnte ich präzise Zeitsteuerungen implementieren. Ich habe auch gelernt, wie Daten zwischen den Tasks ausgetauscht und kommuniziert werden können, was zu einer effizienten Zusammenarbeit zwischen den einzelnen Komponenten führte.

Um Speicherzugriffsprobleme zu vermeiden, verwendete ich Semaphoren, die das Schreiben schützten. So konnte ich sicherstellen, dass keine Datenkonflikte auftraten oder beschädigt wurden. Der SEG\_Driver wurde nicht geschützt, da er nicht von mir stammt und ich ihn nicht ändern wollte.

Ein weiterer Erfolg war die gleichzeitige Ansteuerung des Buzzers aus verschiedenen Tasks, ohne dass es zu Problemen kam. Dies ermöglichte eine synchronisierte Audioausgabe und verbesserte die Benutzererfahrung. Bei jedem Aufruf wird die Zeit des Buzzers aktualisiert, sodass die Audioausgabe bis zum Ablaufen der Zeit fortgesetzt wird und nicht nur einmalig abgespielt wird.

Es ist mir auch gelungen, die Zehntelsekunden in eine 7-Segmentanzeige umzuwandeln. Dies war eine grosse Herausforderung für mich und ich musste mehrere Ansätze ausprobieren, bis ich eine funktionierende Lösung gefunden hatte. Im Internet fand ich keine Lösung, die für meinen Anwendungsfall geeignet war. Durch verschiedene Tests und ständiges Überarbeiten konnte ich eine Lösung finden, die für meine Anwendung geeignet war. So konnte ich eine genaue Darstellung der Zeitmessung realisieren und eine genaue Anzeige der vergangenen Zeit ermöglichen.

Insgesamt habe ich durch dieses Projekt meine Kenntnisse in der Programmierung von Mikrocontrollern und in der Programmiersprache C erheblich erweitert. Ich habe wertvolle Erfahrungen gesammelt und neue Techniken erlernt, die mir auch bei zukünftigen Projekten von Nutzen sein werden. Obwohl ich normalerweise C# benutze und schon lange nicht mehr mit C gearbeitet habe, konnte ich mich schnell wieder einarbeiten und die Sprache effektiv einsetzen.

# Wo waren die Schwierigkeiten?

Anfangs hatte ich Schwierigkeiten mit den Shortcuts der STM32CubeIDE. Die Shortcuts waren ungewohnt oder unpraktisch, was zu einer ineffizienten Arbeitsweise führte. Dies erforderte zusätzlichen Zeitaufwand, um mich mit den Eigenheiten der IDE vertraut zu machen und alternative Arbeitsweisen zu finden. Als Lösung öffnete ich das Projekt in Visual Studio Code und programmierte darin. Nur das Kompilieren und Debuggen habe ich in der STM32CubeIDE durchgeführt. So konnte ich dieses Problem umgehen und die Vorteile von Visual Studio Code nutzen.

Eine weitere Herausforderung war, dass ich mich oft auf die automatische Codegenerierung verlassen musste. Obwohl diese Funktionen hilfreich sein können, führten sie manchmal zu unerwartetem Verhalten und unansehnlichem Code, der nicht zu meinen Komponenten passte. Es war notwendig, den automatisch generierten Code zu überprüfen und anzupassen, um sicherzustellen, dass er den Anforderungen des Projekts entsprach.

Hindernis war für mich, dass die Anforderungen an das Projekt teilweise unklar oder ungenau formuliert waren. Dies führte zu Unsicherheiten und erforderte zusätzliche Kommunikation und Annahmen, um die gewünschten Funktionalitäten zu klären. Die unklaren Anforderungen beeinträchtigten zeitweise die Effizienz und den Fortschritt des Projekts.

# Was könnte schöner gelöst werden?

Eine Möglichkeit, die Durchführung zu verbessern, besteht darin, die Daten für das nächste Mal zu speichern. So könnten z.B. die zuletzt gemessene Zeit und die aktuellen Einstellungen der Stoppuhr und des Timers gespeichert werden. Dies würde verhindern, dass diese Daten bei jedem Neustart des Mikroprozessors verloren gehen, und würde eine nahtlose Fortsetzung der Zeitmessung ermöglichen.

Ein alternativer Ansatz wäre die Berechnung der Zeit ohne die Verwendung von Tasks, sondern basierend auf einer Periode. Anstatt spezifische Tasks für die Stoppuhr und den Timer zu verwenden, könnte eine Methode implementiert werden, die die seit dem Start oder der letzten Aktualisierung verstrichene Zeit berechnet. Dies würde den Ressourcenverbrauch reduzieren und die Implementierung vereinfachen.

Ein weiterer Aspekt, der verbessert werden könnte, ist die Verwendung von Interrupts für die Tastenerkennung. Obwohl Interrupts effizient sein können, um Eingaben in Echtzeit zu erfassen, kann es sein, dass lange Tastendrücke nicht erkannt werden. Eine alternative Lösung könnte die Verwendung von Polling sein, bei dem die Tasten regelmässig abgefragt werden, um die Eingaben zu erfassen.

Eine mögliche Verbesserung besteht darin ein anderes Betriebssystem als FreeRTOS zu verwenden. Dies könnte die Komplexität reduzieren und die Abhängigkeit von externen Bibliotheken beseitigen. Stattdessen könnten einfache Schleifen und Timerfunktionen verwendet werden, um die notwendigen Zeitsteuerungen zu implementieren.

Eine weitere Möglichkeit besteht darin, den Timer nicht über einen separaten Task, sondern über das System zu steuern. Anstatt einen eigenen Task für den Timer zu erstellen, könnte eine zentrale Systemfunktion verwendet werden, die die Zeitmessung und Aktualisierung des Timers übernimmt. Dies würde die Struktur des Codes vereinfachen und eine konsistente Verwaltung der Zeitfunktionen ermöglichen.

Diese Verbesserungen würden das Projekt in einigen Bereichen effizienter, einfacher und flexibler machen. Es wäre möglich, die Daten zu speichern, die Zeitmessung ohne Tasks zu implementieren, eine bessere Tastenerkennung zu ermöglichen, auf FreeRTOS zu verzichten, den Timer über das System zu steuern und ein geeigneteres Tool für die Verlinkung zu wählen.

# Diagramme

