Auslesen, Abbilden und Anzeigen von kontinuierlichen Signalen im Hinblick auf Signale mit hohen Frequenzen

AUTOR: RICHARD WIEDITZ
MATRIKELNUMMER: 10251959

Inhalt

1. Vorwort

2. Programmierung
    1. Vorwort
    2. Signalgenerierung
    3. Signalgewinnung
    4. Representation im Speicher
    5. Diagrammisierung
    6. Echtzeitfähigkeit

3. Problematiken

4. Schlusswort

1. Vorwort

Im Zuge dieser Arbeit wurde ein Programm erstellt.
Das Programm soll ein Signal generieren, simulieren, speichern und darstellen.

An die Punkte Speicherung, Simulation und Darstellung bestehen zusätzlich Echtzeitanforderungen.

Mit Blick auf die Speicherung fungiert das Programm im weiteren Sinne als Schnittstelle für Echtzeitanalyseprogramme.

Die Anwendung trägt den Arbeitstitel ‚Wafer‘.

2.1. Programmierung - Vorwort

‚Wafer' ist ein C Programm. Es basiert auf dem GCC Compiler und verwendet die Bibliothek ‚gnuplot' zur Darstellung eines Signals.

Seit an ‚Wafer' Echtzeitanforderungen gestellt werden verwendet es die betriebssystemspezifische Bibliothek ‚Mach', was verhindert, das ‚Wafer' out-of-the-box eine crossplattformfähige Anwendung ist.

Eine Inbetriebnahme unter einem anderen Betriebssystem als OSX erfordert das Austauschen dieser Bibliothek durch die kongruente Bibliothek des gewünschten Betriebssystems, sowie das Anpassen der Funktionen zur Erhebung der Programmteile in die Echtzeitpriorität.

Um eine zuverlässige Echtzeitfähigkeit zu garantieren müssen die Teilprogramme unter ‚Wafer' zum einen stets in einer vorhersehbaren Zeit abgearbeitet werden und zum anderen effizient genug sein, damit auch hochfrequente Signale in Echtzeit verarbeitet werden können.

2.2. Programmierung - Signalgenerierung

SYNTHESIZER.H

‚Synthesizer' ist ein Teilprogramm von ‚Wafer', welches für die Generierung bzw. die Simulation von Signalen zuständig ist.

Generiert werden kann ein Signal, welches über Parameter im Hauptprogramm eine beliebige Frequenz, Amplitude, Phasenverschiebung und Länge aufweisen kann.

Eine Variante die ‚Synthesizer' bereithält um das Signal zu simulieren ist, das generierte Signal zunächst in eine CSV Datei zu speichern, um es von dort in Endlosschleife auszulesen. Dies soll potenziell die Darstellung von extern generierten Signalen ermöglichen.

Eine andere Variante ist der direkte Aufruf von mathematischen Funktionen aus dem Programm, was die Verwendung eines ADCs nachempfinden soll (Funktion die ihren Rückgabewert über die Zeit verändert).

In beiden Varianten ist die Dauer eines Lesezugriffs über die Samplingrate konfigurierbar, das bedeutet einstellbar auf die Änderungsfrequenz eines beliebigen ADCs.
Das Programm greift dafür auf die Funktionen der ‚Mach' Bibliothek zurück um die Wartezeit nach jedem Lesezugriff zu errechnen und zu warten um die benötigte Zeit auf den ADC einzustellen. Damit bleibt die Zeit für Lesezugriffe konstant O(1) und damit vorhersehbar, was wichtig ist seit an die Simulation Echtzeitanforderungen bestehen.

2.3. Programmierung - Representation im Speicher

WINDOW.H

Das Teilprogramm ‚Window' ist dafür zuständig das Signal im Speicher abzulegen, und bildet die Schnittstelle für Analyseprogramme und Filter.
Es wird ausserdem zur Diagrammisierung verwendet.

Seit die Funktionen zum Einfügten und Lesen des Signals bei hochfrequenten Signalen sehr oft pro Sekunde genutzt werden lag ein besonderes Augenmerk auf der Effizienz dieser Funktionen.

‚Window' reserviert zu diesem Zweck einen statisch allozierten Speicherbereich und verwaltet ihn als Ringbuffer.

Der Ringbuffer bietet lediglich Funktionen zum einfügen und auslesen des Speichers. Zwei parallel laufende Pointer (Start und Endpointer) zeigen Start und Ende des Signals an.

Beim Einfügen eines Wertes werden beide Pointer erhöht bis das Ende des Speicherbereichs erreicht ist. In diesem Fall werden beide Pointer auf den Start des Speicherbereichs zurück gesetzt.
Dies ermöglicht eine sehr geringe Laufzeit für diese Funktion. Sie bleibt in jedem Fall vorhersehbar mit einer Komplexität von O(1).

Die Funktion des Auslesens liefert einen Pointer auf den erste Wert des Signals zurück. Dieser Pointer muss durch iteration das gesamte Signal in Reihenfolge durchlaufen können, was ein Problem darstellt, seit das Signal nicht zwangsläufig in Reihenfolge im Speicher liegt.

Aus diesem Grund wird beim Programmstart der Speicherbereich auf 2N Elementen ausgeweitet, ist also doppelt so groß wie die bei der Programmkonfiguration angegebene Signallänge N zur Speicherung benötigen würde. Im folgenden wird der erste Teil (Speicherstart bis Ntes Element) S1 und der hintere Teil S2 genannt.

2.3. Programmierung - Representation im Speicher

WINDOW.H

Wenn das Signal nun ausgelesen werden soll wird es vom Start von S1 bis zum Endpointer mittels memcpy an den Start von S2 kopiert. Das Signal liegt nun in Reihenfolge im Speicher und kann vom Startpointer bis zum Nten Elemente durchlaufen werden.
Dieser Vorgang ist vorhersehbar mit einer Komplexität von O(1).

Es existiert eine zweite Variante der Auslesefunktion, welche das Auslesen des Signals mit einer konfigurierbaren Auflösung A ermöglicht. Diese ist allerdings gezwungen neuen Speicherbereich zu allozieren und das Signal mindestens in Intervallen zu durchlaufen, weshalb für diese Variante eine allgemeine Komplexität von O(n), genauer $O(A^{-1})$, entsteht (Bsp.: Signallänge: 50000, Auflösung: 1/2000).

2.4. Programmierung - Diagrammisierung

DIAGRAM.H

‚Diagram' ist der Teil des Programms, der für die Diagrammisierung verantwortlich ist. Genutzt wird die ANSI C Bibliothek GnuPlot, welche die Funktionen des Betriebssystems zur Diagrammisierung verwendet. Im Fall von OSX kann das Programm AquaTerm verwendet werden.

GnuPlot muss bei jeder Aktualisierung des Diagramms das gesamte Signal durchlaufen, was der Grund für die Zugriffszeit O(n) ist.

Hierbei ist die Auflösung des Signals konfigurierbar.
Die Funktion zum Zeichnen greift auf Variante Zwei der Auslesefunktion von ‚window' zu, damit auch hochfrequente Signale, genauer Signale, die eine hohe Signallänge zur verlustfreien analyse erfordern, effizient diagrammisiert werden können.

Dabei verliert das Diagram optisch an Informationsgehalt.

Die Aktualisierungsrate ist konfigurierbar. Ein neu zeichnen des Signals dauert damit maximal eine feste Zeitspanne und bleibt damit vorhersehbar.

2.5. Programmierung - Echtzeitfähigkeit

SCHEDULER.H

‚Scheduler' dient dem Zweck das Programm in Echtzeit zu starten.
Dafür benutzt es die ‚Mach' Bibliothek, da ‚Wafer' unter OSX entwickelt wurde.

Um ‚Wafer' unter einer anderen Plattform zu nutzen bedarf es lediglich des Austauschens der betriebssystemspezifischen Funktionen, die aus der ‚Mach' Bibliothek stammen.

‚Scheduler' bietet Funktionen zum einreihen des Programms in die real-time-queue des Betriebssystems, und erlaubt damit präzise Zeitmessungen innerhalb der Funktionen.

3. Problematiken

Leider wurden aus Zeitmangel einige Funktionen nicht bis zur Perfektion ausimplementiert.

Die Echtzeitkritischen Programmteile scheinen zumindest die Priorität angenommen zu haben und erlauben mehr oder minder präzise Zeitmessungen auf, welche aber Schwankungen im hundertstel Millisekundenbereich aufweisen.
Das ist was nicht akzeptabel ist.

Die Ursache für dieses Schwankungen ist bisher ungelöst, möglicherweise besteht ein Problem bei der Konfiguration der Threadpolicy oder die Wartezeitberechnung ist fehlerhaft.

4. Schlusswort

Resümierend kann man sagen, dass im Zuge dieser Arbeit einige kritische Themen bewältigt wurden, das Endziel aber nicht erreicht wurde.
Bis das Programm wirklich in Echtzeit arbeiten kann bedarf es noch einiger Arbeit.

Der Grundstein für ein in den Zielen beschriebenes Programm ist gelegt.

_____

Unterschrift Autor