KONZEPT FÜR EINE GENERISCHE TREIBERSTRUKTUR AUF C++ BASIS FÜR EINGEBETTETE SYSTEME

THOMAS LINNER (BACHELORSTUDIUM INFORMATIK)

Betreuer: Prof. Dr. Florian Künzner, Prof. Dr. Wolfgang Mühlbauer

Einleitung

Vom einzelnen Smartphone bis hin zum Internet of Things (IoT), die Komplexität der entwickelten Produkte nimmt kontinuierlich zu. So werden stets zusätzliche Sensoren und Aktoren integriert, welche zur Erweiterung der bestehenden Funktionalität oder der Überwachung von Geräten und Systemen dienen. So können zum Beispiel Temperatur und Luftfeuchtigkeit innerhalb eines Systems gemessen werden um eventuelle Beschädigungen durch Überhitzung oder externe Einflüsse proaktiv zu erkennen und zu vermeiden. Bei der Auswahl neuer Komponenten fällt die Wahl überwiegend auf digitale und smarte Komponenten. Diese verfügen meist über eine digitale Kommunikationsschnittstelle, über welche sie mit einem Prozessor o. ä. verbunden werden, welcher die notwendige Ansteuerung und Verarbeitung der Daten durchführt. Für diese Kommunikation sind Treiber notwendig, welche das verwendete Kommunikationsprotokoll der Schnittstelle (z. B. Serial Peripheral Interface (SPI) oder Inter-Integrated Circuit (I2C)), sowie die spezifische Ansteuerung der Komponente implementieren. Die Konzeptionierung und Implementierung solcher Treiber stellt ein wichtiges Aufgabengebiet der Softwareentwicklung für eingebettete Systeme dar.

Im Rahmen der Bachelorarbeit wurde eine generische Treiberstruktur für die Ansteuerung zusätzlicher Hardware-Komponenten entwickelt, implementiert und validiert. Diese Struktur soll eine möglichst allgemeingültige Schnittstelle definieren und so eine Standardisierung zukünftiger Treiber fördern.

Grundlagen

Vor der eigentlichen Konzeptionierung und Implementierung der geforderten Treiberstruktur wurde eine Recherche zu den notwendigen Grundlagen durchgeführt. Hierzu zählt unter anderem auch die Definition und der Aufbau eines Treibers. Ein Treiber stellt hierbei eine „Black-Box“ dar, welche die interne Funktionsweise der Komponente durch ein definiertes Application Programming Interface (API) kapselt. Die Anwendungssoftware kann anschließend über bereitgestellte Funktionen dieser Schnittstelle auf die Funktionalität der angebundenen Komponente zugreifen.

Die Hauptaufgabe während der Konzeptionierung einer Treiberstruktur besteht hierbei im Entwurf einer generischen Schnittstelle zur Abstraktion dieser Komponenten. Für eine Kompatibilität mit einer breiteren Auswahl an Komponenten bietet sich eine hierarchische Struktur (Abbildung 1) an. Diese definiert eine allgemeine Schnittstelle für eine abstrakte Kategorie (hier: Sensor), von welcher eine Menge an spezialisierten Schnittstellen für diverse Untergruppen (hier: Temperatur und Beschleunigung) abgeleitet werden. Jede der abgeleiteten Schnittstellen kann hierbei eine erweiterte Funktionalität anbieten, welche nur von dieser Untergruppe unterstützt wird. Die eigentlichen Geräte-Treiber (hier: z. B. Temperatur A) implementieren alle zuvor definierten Funktionen und übernehmen die Kommunikation mit der angebundenen Komponente. Hierbei wird die interne Funktionsweise dieser vollständig durch den Treiber gekapselt.

Auf Basis verschiedener Betriebsmodi der verwendeten Kommunikationsschnittstelle ergeben sich bis zu drei unterschiedliche Implementierungskonzepte, welche im Rahmen der Bachelorarbeit betrachtet wurden:

- Polling
- Interrupt
- Direct Memory Access (DMA)

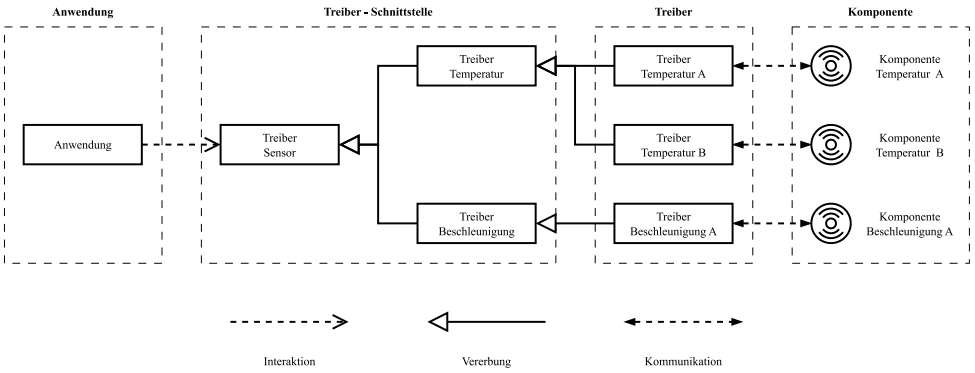

Abbildung 1 Graphische Veranschaulichung einer Treiberstruktur am Beispiel von Sensoren

ROSENHEIMER INFORMATIKPREIS INF-BACHELOR

Zusätzlich zur Recherche über die Definition und den Aufbau von Treibern wurden auch diverse Entwurfsmuster betrachtet, welche für die Konzeptionierung der Treiberstruktur von Interesse waren. Der Fokus lag hierbei auf dem Curiously Recurring Template Pattern (CRTP), welches sich unter anderem mit der Thematik des dynamischen und statischen Polymorphismus befasst. Durch eine Verlagerung der Auflösung des polymorphen Verhaltens von der Laufzeit zur Kompilierzeit, verspricht dieses Entwurfsmuster eine verbesserte Performance bei Funktionsaufrufen einer Klassenstruktur. Diese Aussage konnte anhand einer durchgeführten Laufzeitmessung unter definierten Bedingungen geprüft und bestätigt werden.

Konzeption

Während der Konzeptionsphase wurde anhand von Unified Modeling Language (UML) - Diagrammen eine hierarchische Klassenstruktur – analog zu Abbildung 1 – für die Treiber entworfen. Hierbei wurden unter anderem auch Komponenten betrachtet, welche mehrere Untergruppen (z. B. mehrere Sensortypen) in sich vereinen. Bei einer solchen Struktur sollte unter anderem auch die Thematik der Mehrfachvererbung und dem daraus resultierenden „Diamond“ - Problem betrachtet werden. Einen möglichen Lösungsansatz hierfür bietet zum Beispiel die sogenannte „Composition over Inheritance“. Hierbei werden die gewünschten Funktionalitäten nicht durch Vererbung, sondern durch Referenzen auf andere Objekte, welche diese implementieren, weitergegeben.

Zusätzlich wurde eine Liste von Konzepten untersucht, welche im Rahmen der Treiberentwicklung eine gewisse Relevanz spielen. Diese wurden sowohl theoretisch betrachtet, als auch unterschiedliche Implementierungsansätze analysiert und verglichen. Hierbei wurde beispielweise der Entwicklungsaufwand, die Performance und die Benutzerfreundlichkeit für die Implementierung der Treiber und Anwendungslogik untersucht. Zu diesen Konzepten zählen unter anderem:

- Registerzugriff
- Event-System
- Konfiguration
- Fehlerbehandlung

Implementierung und Validierung

Für eine erste Validierung der konzipierten Treiberstruktur sollte eine exemplarische Implementierung von zwei Geräte-Treibern, sowie eine einfache Beispielanwendung durchgeführt werden. Diese sollten anschließend auf Basis ihres Entwicklungsaufwandes, der Laufzeitperformance, sowie des benötigten Speicherplatzes untersucht and validiert werden.

Bei der Analyse des benötigten Speicherplatzes wurden zwei verschiedene Kategorien untersucht. In einem ersten Schritt wurde die Größe der instanziierten Treiber im Speicher betrachtet. Diese ermöglicht eine Abschätzung des benötigten „Stack“ – Speichers für die Verwendung des Treibers. In einem zweiten Schritt wurde die Größe des kompilierten Programmcodes des Treibers ermittelt. Diese ermöglicht eine Abschätzung des benötigten Random Access Memory (RAM) und Read Only Memory (ROM) auf dem Microcontroller.

Bei der Laufzeit-Analyse der einzelnen Treiber-Funktionen wurden zwei unterschiedliche Verfahren verwendet und verglichen. Für die erste Methode wurde ein Debugger der Firma „Lauterbach“, sowie die Software „TRACE32“ verwendet. Diese erlaubt unter anderem eine Auswertung der prozentualen Laufzeit aller gemessenen Funktionen. Für die zweite Methode wurden diverse Signale (z. B. Kommunikationsprotokoll) mit einem Oszilloskop gemessen. Anhand dieser wurden die Laufzeiten für verschiedene Zeitbereiche (z. B. IDLE) bestimmt.

Fazit

Zum Abschluss der Bachelorarbeit wurden alle zuvor gesammelten Ergebnisse in einem finalen Fazit zusammengefasst. Durch die Implementierung der Beispieltreiber konnte validiert werden, dass die entwickelte Treiberstruktur sowohl einfache, als auch komplexere, Treiber unterstützt. Der Entwicklungsaufwand für einen einfachen Geräte-Treiber belief sich hierbei auf ca. ein bis zwei Arbeitstage. Die Analyse des benötigten Speicherplatzes hat ergeben, dass selbst bei kleineren Microcontrollern noch ausreichend Speicherplatz für die eigentliche Anwendung zur Verfügung steht. Auch die Laufzeit-Analyse hat ergeben, dass noch genügend IDLE - Zeit für die Anwendungslogik verfügbar ist. Bei der Gegenüberstellung der beiden verwendeten Varianten für die Laufzeit-Analyse konnte festgestellt werden, dass sich die Messergebnisse bis auf geringe Abweichungen decken, siehe Abbildung 2.

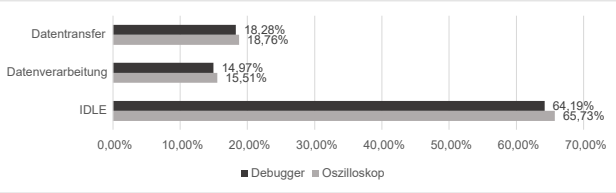

Abbildung 2 Ergebnisse der Laufzeit-Analyse