Evaluation

Inhaltsverzeichnis

Evaluation

Die in den Anforderungen definierten Ziele konnten vollständig umgesetzt werden. Durch die Windows- und Firmware-Tests, sowie das Erstellen der Benchmark-Programme, konnte die grundsätzliche Funktionsfähigkeit der MP-VM und Anbindung bestätigt werden. Auch alle in den Reviews besprochenen bzw. gewünschten Änderungen an der ORB-Firmware, konnten umgesetzt werden. Daher ist im Hinblick auf diese Punkte, diese Arbeit seitens technischer Anforderungen ein voller Erfolg. Das Benchmarking konnte auch die MP-VM-Performance der C++-Performance gegenüberstellen. Es gibt einige Anwendungen, die durch die Verwendung der MP-VM deutlich langsamer werden oder unter Umständen gar nicht korrekt funktionieren könnten. Dies war jedoch zu erwarten. Die Verwendung eines Interpreters wird in jedem Fall langsamer als eine C++-Implementation laufen. Es konnte jedoch auch gezeigt werden, dass es Anwendungen gibt, welche durch das Verwenden der MP-VM nicht deutlich eingeschränkt werden. So wird es viele Anwendungen geben, welche problemlos ihre Funktionen erfüllen können. Die verschiedenen Benchmark-Programme wurden miteinander verglichen und als Gesamtheit betrachtet. Abschließend wurden die Ergebnisse zusammengeführt und eine entsprechende Schlussfolgerung gezogen. Diese findet sich im Implementierungsreport unter 1.6.5. Schlussfolgerung. Im Laufe der Implementierung konnten viele Probleme identifiziert und gelöst werden. Hier ist anzumerken, dass es ein paar Probleme gibt, deren Lösung den Umfang dieser Bachelorarbeit überschritten hätten, wie z.B. das Einbinden von Middleware über ein File-System. Für diese auftretenden Probleme wurde jedoch in jedem Fall eine Lösung gefunden, welche eine stabile Verwendung der ORB-Firmware gewähreisten kann. Es wird auch mit solchen Problemen umgegangen, die nicht unbedingt lösbar sind, wie der Umgang mit Hard-Faults, welche durch die MP-VM produziert werden können. Probleme, welche nicht zur Gänze gelöst werden konnten, werden im Ausblick weiter ausgeführt. Ebenso weitere Verbesserungsvorschläge, die diese Arbeit noch verfeinern könnten oder auch in einer Anschluss-Arbeit umgesetzt werden könnten. Zusammengefasst wurden alle Anforderungen erfüllt. Es liegt eine stabile ORB-Firmware mit den gewünschten Funktionen und API-Dokumentationen vor. Die ORB-Application wird durch die MicroPython-VM vollständig abgebildet. Die Programmierung mit MicroPython ist konzeptionell mit der Programmierung in C++ gleichzustellen. Es wird Middleware und das Verwenden des ORB-Monitors unterstützt.

Ausblick

Im Ausblick werden weitere angedachte Verbesserungsmöglichkeiten vorgestellt.

Performance Verbesserung

Die MP-VM hat im Vergleich zu der C++-Application je nach Anwendung eine vergleichbar schlechte Performance. Gerade solche Funktionen welche die Get-Funtion für Sensor- und Motor-Daten verwenden. Hier wird im Moment ein Dictionary aufgebaut. Dies ist gewählt worden, da es sich um einen bereits in MicroPython integrierten Typ handelt. Hier kann jedoch vermutet werden, das ein Dictionary zu mächtig für diesen Anwendungsfall ist. So könnte man versuchen Sensor- und Motor-Report-MicroPython-Objekte anzulegen. Diese würden dann für genau diesen Zweck umgesetzt sein. Diese könnten den Overhead, der mit dem komplexen Dictionary-Objekt zusammenhängt, minimieren. Hier handelt es sich jedoch eher um eine Vermutung, als eine Anleitung die Performance zu verbessern. Die Annahme stützt sich darauf, dass ein spezialisiertes Objekt besser optimiert werden kann. Es können so z.B. Typ-Checks umgangen werden, wenn der Nutzer nicht in der Lage ist dieses Objekt zu erstellen. Das Sensor- oder Motor-Report-Objekt würde also nur durch die Rückgabe einer Funktion instanziiert werden können. Eine weitere Übelegung, wäre das Verwenden einer anderen MicroPython-Objekt repräsentation. Hier müssten die Vor- und Nachteile der MicroPython-Objekt-Repräsentationen gründlich überlegt werden. Wie bereits, in Konzepte: Wie funktioniert die Typ-Zuordnung (Konkrete-Typen), erwähnt wird im Moment 'MICROPY_OBJ_REPR_A' verwendet. Die anderen MicroPython-Objekt-Repräsentation Funktionieren konzeptionell gleich. Die Modul-Implementationen müssten nicht angepasst werden. Je nach gewählter Repräsentation könnte ein größerer Heap-Speicherverbrauch zugunsten einer besseren Performance in Kauf genommen werden.

File-System in die ORB-Firmware Integrieren

Das Integrieren eines File-Systems in die ORB-Firmware wäre eine Verbesserung die man vornehmen könnte. Diese bringt, zum aktuellen Stand Probleme mit sich. Der Flash-Speicher des ORB ist sehr limitiert. Daher wurde zu diesem Zeitpunkt davon abgesehen. Jedoch würde die Integration Vorteile mit sich bringen. Python-Programme könnten auf dem File-System basierend eine Import-Struktur verwenden. Dies würde Nutzern erlauben MicroPython für das ORB zu schreiben, ohne die '.build'-Datei zu benötigen. Import-Abhängigkeiten wären so klarer strukturierbar. Um dies umzusetzen, muss klar bestimmt werden wie viel Overhead durch ein File-System entsteht. In diesem Schritt wäre es möglich zu überlegen, das ORB um einen externen Speicher zu erweitern. Hier müsste zu allererst überlegt werden, welches Speicher-Medium gewählt werden soll. So könnte ein externer Flash-Speicher fest auf das Board integriert werden. Falls diese Änderung in Betracht gezogen wird, so muss sichergestellt werden, dass auch das C++-Programm von dem Speicher-Medium aus ausgeführt werden kann.

Peripherie durch Middleware abbilden

Es gibt eine Vielzahl an Peripherie, welche durch das ORB unterstützt wird. In diesem Projekt wird durch 'Build'-Dateien eine Möglichkeit geboten, mehrere Python-Scripts zu einem Programm zu kompilieren. Dieser Prozess kann genutzt werden um sogenannte Middleware umzusetzen. Dies ist ein durch die ORB-C++-Programme eingeführtes Konzept. Es sollen zusätzliche Klassen angeboten werden, welche verschiedene Peripherien für eine nutzerfreundliche Verwendung bereitstellen. So sollten auch für die Python-Umgebung zusätzliche Klassen und Module umgesetzt und als eine Middleware angeboten werden.

Code-Editor in ORB-Monitor integrieren

Durch die Erweiterung der ORB-Firmware um die MP-VM wurde ein Build-Prozess für die Python-Programme mithilfe des ORB-Util umgesetzt. Dies ist jedoch ein für dieses Projekt spezifischer Prozess. Ein Nutzer muss sich mit dem Command-Line-Tool vertraut machen um Python-Programme zu programmieren. Das ORB-Util ist zwar ein nutzerfreudnliches Tool, aber man könnte diesen Vorgang noch verbessern. Eine Möglichkeit wäre das Integrieren eines Code-Editors in den ORB-Monitor. Dieser könnte einfach das ORB-Util zum Kompilieren von Python-Programmen nutzen. Die Projekt-Struktur, also im Prinzip die `.build`-Datei, könnte dabei durch den ORB-Monitor abgebildet und generiert werden. Der Nutzer müsste hier im besten Fall die Möglichkeit haben, Python-Dateien anzulegen und diese in Ordnern ablegen zu können. Im besten Fall hätte die Editor-Oberfläche eine Python-Syntax-Highlighting-Unterstützung. Durch die Integration eines Code-Editors in die ORB-Firmware wäre es für einen Nutzer möglich, Python-Programme für das ORB zu schreiben, ohne ein zustätzliches Tool zu lernen. Hier könnte man sich von MicroPython-Editoren wie zum Beispiel 'Thony' inspirieren lassen. Dadurch das die MP-VM auch unter Windows getestet werden kann, könnte man hier auch das Debuggen unter Windows anbieten.

Ausgabe von Print Anweisungen und Fehler-Meldungen

Die Ausgaben von Print-Statements und Error-Stacktraces stellten sich als eine Herausforderung dar. Die Schwierigkeit Micropython-Errors auszugeben, liegt an durch den ORB-Monitor vorgegebenen Limitierungen. Hier ist sowohl die Länge als auch die Formatierung der Ausgabe ein Problem. Wie bereits im Implementierungsreport beschrieben, ist die MP-VM-Print-Funktion darauf ausgelegt Strings so auszugeben, wie man es über UART erwarten könnte. So werden u.A. die Ausgabe von Arrays also z.B. `"[1,2,3,4,5]"` dadurch realisiert, dass das Array als Einzelteile ausgegeben wird. So bekommt man von der Micropython Print-Funktion, in diesem Fall `"[" "1, " "2, " "3, " "4, " "5, " "]\r\n"` also alle Einzelteile der Ausgabe, als einen eigenen String übergeben. Dieses Problem würde sich durch eine UART-ähnliche Ausgabe über den ORB-Monitor lösen. Also durch eine wachsende oder scrollbare Text-Box und nicht etwas die 4 vorgegebenen Zeilen. Dadurch wäre es auch möglich größere Ausgaben wie die von Fehler-Meldungen mit den zusätzlichen Informationen über den Stacktrace zu ermöglichen. Der ORB-Monitor würde bei dieser Lösung die Print-Ausgabe wie durch MicroPython vorgegeben bekommen. Alternativ könnte man die Micropython-VM Fehler-Meldung im Flash-Speicher ablegen und von dem ORB-Monitor als eine Text-Datei auslesbar machen. Die Anpassung des ORB-Monitor-Text-Feldes als Lösung erscheint jedoch am nutzerfreundlichsten.

MP-VM kann Hard-Faults produzieren

Durch die MP-VM ist in die ORB-Firmware eine Komponente integriert worden, welche Speicher-Zugriffs-Fehler produzieren kann. Die MP-VM macht in einigen Fällen nicht ausreichende Überprüfungen ob, zum Beispiel, noch genügend MP-VM-Heap-Speicher zur Verfügung steht. Um mit diesem Problem umzugehen wurde ein Fault-Handler in die ORB-Firmware aufgenommen. Dies ist jedoch ein Problem, welches vermutlich nicht zu lösen ist. Denn für diesen Zweck müsste das MicroPython-Projekt vollständig überarbeitet werden. Es müssten an allen Stellen, in denen ein Objekt in dem Heap-Speicher angelegt oder verändert wird, zusätzliche Überprüfungen eingebaut werden. Dies wäre ein massiver Aufwand und würde die Performance der MP-VM stark negativ beeinflussen. Falls der Nutzer auf diese Probleme trifft, kann er jedoch selbst Garbage-Collect-Zyklen in sein Projekt einbauen. Der beste Umgang mit diesem Problem ist wahrscheinlich ein regelmäßiges Re-Basen des MicroPython-Submodules auf dem aktuellen Release-Branch des MicroPython-Projektes. So könnte das ORB-Projekt durch neue Bugfixes und Performance-Verbesserungen des MicroPython-Projektes Profitieren.

Integration von Threading

Die Integration von Funktionen, welche das Verwenden von Threads erlauben, wären von Vorteil. Hier könnte man sich an bereits bestehenden MicroPython-Ports orientieren. Es gibt MicroPython-Ports wie zum Beispiel den Raspberry-Pi-Pico-Port. Diese Ports bieten eine einfache Variante von Threads an. So kann der Nutzer nicht selber Threads erstellen. Es wird eine statische Anzahl an Threads unterstützt. Der Nutzer kann den Threads nur Funktionen zuweisen, diese starten und stoppen. Die Integration solcher einfachen Threads in die ORB-C++-Application sollte dabei relativ gradlinig sein. Hier könnte man pro angebotenen Thread eine Task erstellen. Hier würde dem Nutzer die Möglichkeit gegeben werden, die Update-Funktion eines Threads zuzuweisen. Die `orblocal.h` würde damit zum Beispiel um folgendes erweitert werden:

```
void threadStart(int id);
void threadStop(int id);
```

```
void threadSetFunction(int id, void* fun);
```

Es muss jedoch auch beachtet werden, dass in den ORB-Funktionen Race-Conditions berücksichtigt und entsprechend behandelt werden. Wie zum Beispiel in der Set-Motor-Funktion. Ohne weitere Anpassungen würde hier eine Race-Condition auftreten, da auf eine gemeinsame Motor-Task geschrieben wird. Je nach Reihenfolge der Operation könnte so ein ungewollter Motor-Zustand entstehen. Die Integration von Threading wird von der MicroPython-VM unterstützt. Für diesen Zweck müssen für das Threading benötigte Funktionen umgesetzt werden. Diese werden von der MicroPython-VM vorgegeben. Ein guter Einstiegspunkt wäre hier das Umsetzen einer 'mpthreadport.h'-Datei. Hier wird eine Vielzahl an für das Threading benötigten Funktionen der MicroPython-VM erwartet. Hier sind vermutlich noch weitere Änderungen notwendig. So muss auch die ORB-Firmware und MicroPython-Konfiguration angepasst werden.

> Für den Raspberry-Pi-Pico-Port findet sich diese Datei unter: micropython/ports/rp2/mpthreadport.h.

Da alle Threads für die MicroPython-VM eine geteilte VM haben, müsste die ORB-Firmware hier um Konzepte, wie zum Beispiel Mutex, erweitert werden. Dies ist jedoch mit erheblichem Implementierungsaufwand verbunden. Es kann sich an der Vielzahl an bestehenden MicroPython-Ports orientiert werden.