Informatik 11
Embedded Software

RWTH AACHEN UNIVERSITY

Praktikum Systemprogrammierung

Versuch 4

Externer Speicher

Lehrstuhl Informatik 11 - RWTH Aachen

26. Oktober 2020

Inhaltsverzeichnis

Dieses Dokument ist Teil der begleitenden Unterlagen zum *Praktikum Systemprogrammierung*. Alle zu diesem Praktikum benötigten Unterlagen stehen im Moodle-Lernraum unter https://moodle.rwth-aachen.de zum Download bereit. Folgende E-Mail-Adresse ist für Kritik, Anregungen oder Verbesserungsvorschläge verfügbar:

support.psp@embedded.rwth-aachen.de

4 Externer Speicher

Die in den vorherigen Versuchen implementierten Komponenten unseres Betriebssystems beanspruchen bereits einen Großteil des internen SRAMs auf dem Mikrocontroller. Nur noch etwa 1 KiB verbleibt zur freien Verwendung von Programmen als Heap-Speicher. Um dennoch dynamische Speicheranforderungen in realistischem Umfang erfüllen zu können, erweitern wir in diesem Versuch das Evaluationsboard um einen zusätzlichen externen Speicherbaustein.
Zum Einsatz kommt der Speicherbaustein 23LC1024 der Firma Microchip, welcher 128 KiB als SRAM bereitstellt. Der Speicherbaustein wird über das Serial Peripheral Interface (SPI) mit dem ATmega 644 verbunden. Dieses Bussystem ist in eingebetteten Systemen weit verbreitet und wird häufig zur Einbindung externer Peripherie eingesetzt. Eine darauf aufbauende Protokollschicht wird es dem Betriebssystem ermöglichen, den Speicher effizient zu beschreiben und zu lesen. Als Resultat wird die SPOS und den Prozessen zur Verfügung stehende Speicherkapazität deutlich erhöht.

4.1 Versuchsinhalte

Nachfolgend werden zunächst die Grundlagen der SPI-Kommunikation eingeführt und die Integration des ATmega 644 mit dem 23LC1024 erläutert. Anschließend wird eine Treiberschicht für SPI implementiert. Darauf aufbauend wird das Protokoll des 23LC1024 als Grundlage eines neuen Speichertreibers umgesetzt.
Des Weiteren wird die abstrakte Heap-Implementierung um weitere Allokationsstrategien sowie die Möglichkeit zur dynamischen Anpassung bereits existenter Speicherbereiche erweitert. Ebenso wird die automatische Speicherbereinigung in diesem Versuch angepasst, um effizient mit der deutlich erhöhten Speicherkapazität umzugehen.

Hinweis

Es wird empfohlen evtl. noch unabgeschlossene Präsenzaufgaben/Testtasks aus dem vorherigen Versuch jetzt zu vervollständigen.

4.2 Lernziel

Das Lernziel dieses Versuchs ist das Verständnis der folgenden Zusammenhänge:

- Spezifikation des Serial Peripheral Interface
- Selbständiger Umgang mit Datenblättern
- Protokoll des 23LC1024
- Größenänderung dynamischer Speicherbereiche
- Weitere Strategien zur Allokation von Speicher

4.3 Grundlagen

Die im Folgenden geschilderten Grundlagen behandeln zunächst das Serial Peripheral Interface im Allgemeinen und anschließend dessen Implementierung auf dem ATmega 644. Auf dem SPI aufbauend verwendet der 23LC1024 ein eigenes Kommunikationsprotokoll, welches Sie sich eigenständig mit Hilfe des Datenblattes aneignen.
Zuletzt wird auf die Theorie der neuen Funktionen der Heap-Implementierung eingegangen.

4.3.1 Serial Peripheral Interface (SPI)

SPI ist ein im Umgang mit Mikrocontrollern gängiges Bussystem für den seriellen Datenaustausch, welches die Anbindung externer Peripherie (z.B. Sensoren, Aktoren, Speicher) an einen Hauptprozessor ermöglicht. Als Bussystem teilen sich die Kommunikationsteilnehmer einen gemeinsamen Übertragungsweg, wobei im Fall von SPI ein Master die Buskommunikation steuert und die sogenannten Slaves nur auf Anfrage des Masters kommunizieren dürfen. Das SPI erlaubt es Daten zwischen Master und jeweils einem der Slaves in Form von Bytes auszutauschen, die semantische Interpretation der ausgetauschten Bytes ist jedoch Abhängig vom Anwendungsfall und der externen Peripherie. Diese anwendungsspezisiche Weiterverarbeitung der Nutzdaten muss auf höheren Protokollschichten erfolgen. Prinzipiell unterstützt SPI eine beliebige Anzahl an Slaves, in diesem Versuch ist jedoch der Speicherbaustein der einzige Slave.
Wie in Abbildung 4.1 dargestellt, werden alle Busteilnehmer über gemeinsame Steuerleitungen verbunden: Clock (CLK), Master-out-Slave-in (MOSI) und Master-in-Slave-out (MISO). Jeder Slave erhält zusätzlich eine exklusive Leitung zum Master — Chip Select (CS) — über die der Master auswählt, mit welchem Slave er kommunizieren möchte.
Der Chip Select ist in der Regel *active-low* (notiert durch $\overline{\mathsf{CS}}$), d.h. ein niedriges Spannungspotential (GND) aktiviert den Slave.
Bei SPI handelt es sich um ein serielles und synchrones Bussystem. Das bedeutet, dass einzelne Bits nacheinander gesendet werden und die Übertragung mittels der Clock-Leitung synchronisiert wird. Der Master sendet über MOSI seine Daten an den Slave

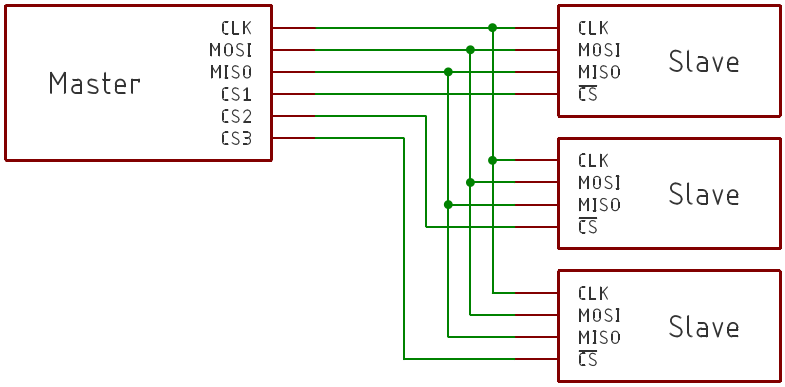

Abbildung 4.1: Steuerleitungen beim SPI mit 3 Slave-Geräten.

und der Slave sendet seine Daten an den Master zeitgleich über MISO. Hierzu verändern Master und Slave das Spannungsniveau an den Datenleitungen MISO und MOSI. Eine Besonderheit des SPI ist die Tatsache, dass während einer Übertragung stets zeitgleich von beiden Kommunikationsteilnehmern Daten übermittelt werden. Dies ist je nach Anwendung nicht immer sinnvoll. In solchen Fällen müssen die Daten in darüberliegenden Protokollschichten verworfen werden.

Hinweis

Die Bezeichner der Steuerleitungen (CLK, MISO, MOSI) können zwischen Datenblättern variieren. Die Funktionsweise ist allerdings identisch und die entsprechende Zuordnung geht aus dem Kontext des jeweiligen Datenblattes hervor.

Clock Timing Das Clock-Signal ist ein Rechtecksignal, welches vom Master erzeugt wird und die Taktfrequenz des Busses vorgibt. Die Datenleitungen werden stets bei einem Pegelwechsel der Clock, auch Signalflanke genannt, ausgelesen. Welche Signalflanke dabei genau betrachtet wird, gibt der verwendete **SPI-Modus** vor.
Es wird zwischen vier SPI-Modi unterschieden. Diese ergeben sich aus zwei binären Einstellungen zur Polarität und Phasenlage. Die Polarität bestimmt den neutralen Signalzustand der Clock-Leitung. Die Phasenlage stellt ein, ob bei der ersten Flanke (*Leading Edge*) oder zweiten Flanke (*Trailing Edge*) ausgelesen wird. Tabelle 4.2 fasst diese Einstellungen noch einmal zusammen.

SPI-Modus	Polarität	Phasenlage
0	`CPOL=0`, Idle Low, Active High	`CPHA=0`, Leading Edge
1	`CPOL=0`, Idle Low, Active High	`CPHA=1`, Trailing Edge
2	`CPOL=1`, Idle High, Active Low	`CPHA=0`, Leading Edge
3	`CPOL=1`, Idle High, Active Low	`CPHA=1`, Trailing Edge

Abbildung 4.2: SPI-Clock-Timing

Bitreihenfolge Beim SPI wird in jedem Kommunikationszyklus ein Byte übertragen. Zur korrekten Interpretation der einzelnen übertragenen Bits muss zuletzt noch spezifiziert werden, ob es sich beim zeitlich zuerst gesendete Bit um das *Least-Significant-Bit* (LSB) oder das *Most-Significant-Bit* (MSB) handelt. Dieser Parameter wird als *Data Order* bezeichnet.

4.3.2 Beispiel

Abbildung 4.3 zeigt beispielhaft eine SPI-Übertragung mit folgender Konfiguration:

- `CPOL=0`
- `CPHA=0`
- Data Order: LSB first

In zeitlicher Reihenfolge (von links nach rechts im Diagramm) wird auf der MISO Leitung `01100100` gesendet. Der Slave überträgt also das Byte `0x26` an den Master.

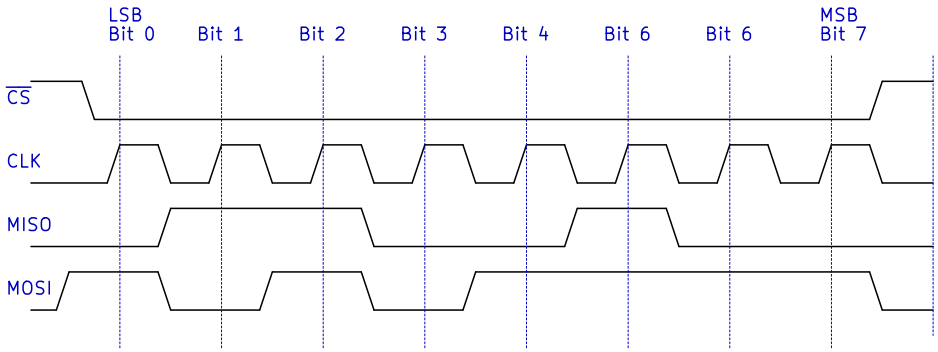

Abbildung 4.3: SPI-Beispiel Übertragung.

Lernerfolgsfragen

- Was überträgt der Master an den Slave (Abbildung 4.3)?
- Welche Bytes werden übertragen, wenn mit Data Order *MSB first* interpretiert wird?
- Wie sehen die Datenleitungen (MISO, MOSI) aus, wenn `CPHA=1`? Am Ende des Dokuments stehen leere Diagramme zum Experimentieren bereit (Abbildung 4.13).

4.3.3 SPI-Modul im ATmega 644

Um den Umgang mit dem SPI-Bus effizient zu gestalten, verfügt der ATmega 644 über ein dediziertes SPI-Modul. Die Busleitungen müssen nicht durch den Entwickler direkt über die I/O-Register angesteuert werden, dies wird vom integrierten SPI-Modul übernommen.
Das SPI-Modul im ATmega 644 kontolliert selbständig vier Pins an Port B. Zum einen sind dies die drei geteilten Busleitungen MISO, MOSI und CLK. Zusätzlich besitzt der ATmega 644 auch selber einen CS welcher durch das SPI-Modul kontrolliert wird. Dieser ist relevant, falls man den Mikrocontroller nicht als Master, sondern als Slave nutzen möchte. Die Beschaltung fremder CS-Leitungen hingegen wird nicht vom SPI-Modul übernommen sondern muss manuell durch die I/O-Register erfolgen. Die CS-Leitung des 23LC1024 ist mit Pin 4 an Port B verbunden (siehe Tabelle 4.7).
Im Folgenden werden diese Grundlagen zur Verwendung des SPI-Moduls erläutert. Weitere Details lassen sich dem Datenblatt des ATmega 644 entnehmen (Abschnitt 16. SPI, Seite 154-163).

Das SPI-Modul wird über drei Register angesteuert:

- SPI Control Register (`SPCR`)
- SPI Status Register (`SPSR`)
- SPI Data Register (`SPDR`)

Die beiden ersten Register dienen zur Konfiguration und das dritte Register ist für die eigentliche Datenübertragung verantwortlich. Die Bitposition der im Folgenden erläuterten Optionen ist in Tabelle 4.4 dargestellt.
Das Setzen der Bits `SPE` (SPI Enable) und `MSTR` (Master) aktiviert das SPI-Modul und konfiguriert den AVR als Bus-Master.

Register/Bit	7	6	5	4	3	2	1	0
Control Register	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
Status Register	SPIF	WCOL	-	-	-	-	-	SPI2X

Abbildung 4.4: Aufbau der SPI-Register

SPIE (SPI-Interrupt Enable) aktiviert die dem SPI-Modul zugehörige ISR, welche im Rahmen des Praktikums allerdings nicht verwendet wird.

DORD (Dataorder) bestimmt die Bitreihenfolge der Übertragung. Wenn DORD auf 0 gesetzt ist, wird das MSB zuerst übertragen. Entsprechend wird bei DORD=1 das LSB zuerst übertragen.

CPOL und CPHA verhalten sich analog zu den in Abschnitt 4.3.1 erläuterten Grundlagen.

SPR1, SPR0 und SPI2X (Clock Rate Select und Double Speed) konfigurieren die Taktfrequenz des SPI-Moduls auf Basis zur CPU Frequenz (*Prescaling*). Die konkreten Taktraten können Tabelle 4.5 entnommen werden.

SPI2X	SPR1	SPR0	SPI-Frequenz
0	0	0	$f_{CPU}/4$
0	0	1	$f_{CPU}/16$
0	1	0	$f_{CPU}/64$
0	1	1	$f_{CPU}/128$
1	0	0	$f_{CPU}/2$
1	0	1	$f_{CPU}/8$
1	1	0	$f_{CPU}/32$
1	1	1	$f_{CPU}/64$

Abbildung 4.5: SPI-Taktfrequenz

Der AVR überträgt SPI-Daten byteweise. Um ein Byte zu übertragen wird es in SPDR geschrieben. Die Übertragung wird dann automatisch durchgeführt und das im Austausch empfangene Byte kann nach Abschluss der Übertragung ebenfalls aus SPDR gelesen werden. Der Abschluss der Übertragung wird durch ein gesetztes SPI-Interrupt Flag SPIF im SPI Status Register signalisiert. Auch bei deaktiviertem Interrupt wird diese Bit durch das SPI-Modul gesetzt.

Das WCOL-Bit ist für unsere Implementierung nicht von Bedeutung.

Achtung

Falls der CS des ATmega 644 als Input konfiguriert ist und an ihm ein niedriges Spannungspotenzial (GND) anliegt, so versetzt der Mikrocontroller sich automatisch in den Slave Modus. (Datenblatt, 16.3 – $\overline{\text{SS}}$ Pin Functionality)

4.3.4 SRAM-Baustein

Der 23LC1024 Speicherbaustein verfügt über 1 MBit bzw. 128 KiB Speicher, welcher byteweise, sequentiell und abschnittsweise über den SPI-Bus beschrieben und ausgelesen werden kann. Lesen Sie das Datenblatt des 23LC1024, welches im Moodle-Lernraum für Sie bereitsteht. Identifizieren zunächst Sie die Bezeichnung der Busleitungen. Leiten Sie anschließend aus den Timing-Diagrammen den verwendeten SPI-Modus ab. Machen Sie sich mit dem verwendeten Protokoll zum Schreiben und Lesen von Daten und Setzen von Einstellungen vertraut.

Lernerfolgsfragen

- Welche Befehle können an den 23LC1024 gesendet werden?
- Welche davon müssen Sie verwenden?
- Welche weiteren Daten werden bei welchen Befehlen übertragen?
- Welche maximale SPI-Taktfrequenz unterstützt der 23LC1024?
- Ist die CS-Leitung in unserer Implementierung notwendig, obwohl nur ein Slave angesteuert wird?

Achtung

Die zur Implementierung relvanten Inhalte des Datenblatts zum 23LC1024 von Microchip sind verpflichtender Bestandteil der Nachbefragung am Ende des Versuchs.

4.3.5 Speichererweiterungsplatine

Die in Abbildung 4.6 dargestellte Erweiterungsplatine verbindet den 23LC1024 mit einer acht-poligen Buchsenleiste, durch welche die Platine auf das Evaluationsboard an Port B aufgesteckt werden kann. Dadurch werden die Busleitungen des SPI verbunden (vgl. Abschnitt 4.7). Das weiße Dreieck markiert Pin 1, welcher die Orientierung in Bezug auf das Evaluationsboard eindeutig festlegt. Des Weiteren muss die Spannungsversorgung über ein zweipoliges Kabel am Evaluationsboard angeschlossen werden.

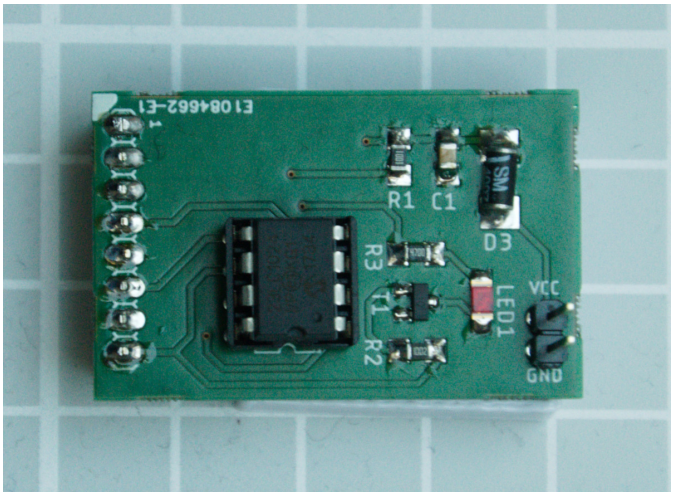

Abbildung 4.6: Erweiterungsplatine mit 23LC1024

4.3.6 Dynamische Speicheranpassung

In Versuch 3 wurden zwei wichtige Funktionen zur Verwaltung dynamischen Speichers implementiert: `os_malloc` und `os_free`. Es gibt jedoch Anwendungsfälle, bei denen die Größe eines allozierter Speicherbereichs angepasst werden muss. Prinzipiell könnten in solchen Fällen verkettete Listen oder ähnliche Datenstrukturen verwendet werden, um Datenmengen variabler Größe zu verwalten.
Hierbei gibt es zwei wesentliche Probleme: Zunächst geht bei diesen Datenstrukturen die zentrale Eigenschaft des wahlfreien Zugriffs (in konstanter Zeit) des verwendeten Speichers verloren. Zusätzlich ist der Overhead (Speicherverbrauch für die Verwaltungsdatenstrukturen) bei vielen Heap-Implementierungen konstant, anstatt wie bei SPOS linear in der Größe eines Speicherbereichs, sodass viele kleine Speicherbereiche ineffizienter sind als ein Großer. Hinzu kommt der Overhead der Datenstrukturen selbst, welche pro Element mindestens einen Zeiger benötigen.
Diese Probleme können gelöst werden, indem bereits existierende Speicherbereiche dynamisch vergrößert und verkleinert werden, was als *Reallokation* bezeichnet wird. Außerhalb der Speicherverwaltung kann dies nur geschehen, indem ein neuer (passender) Speicherbereich mit `os_malloc` angefordert, alle Daten in den neuen Bereich kopiert und

der alte Bereich mit `os_free` wieder freigegeben wird.
Dieser Ansatz ist aber problematisch, da der restliche Speicher zu klein sein könnte, um den neuen Bereich aufzunehmen, obwohl der freie Speicherabschnitt nach Entfernen des ursprünglichen Bereichs ausreichen würde. Dies ist in Abbildung 4.7 für die Reallokation mit `newSize = 5` dargestellt.

Abbildung 4.7: Fragmentierung führt bei Reallokation des blauen Bereichs mit Größe 5 zu falschem Verhalten bei Verwendung des obigen Ansatzes.

Außerdem ist das Verschieben der Daten vergleichsweise aufwändig. Falls, wie im Beispiel in Abbildung 4.7, der alte Speicherbereich in einem Abschnitt liegt, der auch den neuen Speicherbereich beinhalten könnte, so können die Daten an der bisherigen Adresse verbleiben, anstatt verschoben zu werden. Für genaue Effizienzbetrachtungen – die Untersuchung der Gesamtlaufzeit – wäre selbstverständlich zusätzlich die Laufzeit der `os_malloc` und `os_free` Funktionen zur Gesamtlaufzeit hinzuzuzählen.
Falls die Reallokation von der Speicherverwaltung des Betriebssystems durchgeführt wird, ist diese von den hier beschriebenen Effekten, je nach Implementierung, nicht betroffen, da diese Zugriff auf die gesamte Struktur des Heaps (bei SPOS auch der Heapmap) hat. Hierbei kann effizient geprüft werden, ob der Speicherbereich erweiterbar ist, oder ob ein neuer Bereich angefordert werden muss.

Lernerfolgsfragen

- Welche Probleme treten auf, wenn die Reallokation durch Anwendungsprozesse implementiert wird, anstatt von der betriebssysteminternen Speicherverwaltung? Betreffen diese Probleme die Korrektheit, die Effizienz oder beides?
- Unter welcher Bedingung ist eine effiziente Vergrößerung eines Speicherbereichs nicht möglich?
- Gibt es ein Beispiel, bei dem das Verschieben der Daten notwendig ist, obwohl insgesamt nur ein einziger Speicherbereich existiert? Wie sieht dies aus?

4.3.7 Allokationsstrategien

Die Platzierung von neu angeforderten Speicherbereichen innerhalb des Heaps kann auf unterschiedliche Arten bestimmt werden. In Versuch 3 wurde bereits die Allokationsstrategie *First-Fit* vorgestellt. Im Folgenden werden in Ergänzung dazu drei weitere Allokationsstrategien vorgestellt. Zusätzlich wird jede dieser Strategien am Beispiel eines konkreten Speicherbereichs mit einer Größe von 25 Byte veranschaulicht werden. In

diesem sind 9 Byte in Form von 6 Bereichen belegt (gelb) und noch 16 Byte frei (weiß).

Abbildung 4.8: Fragmentierter Speicher mit 25 Byte Größe

Next-Fit

Die Allokationsstrategie *Next-Fit* bzw. *Rotating-First-Fit* sucht analog zu *First-Fit* den ersten passenden Speicherbereich. Die Suche nach einem freien Speicherbereich mit ausreichender Größe beginnt jedoch bei der ersten Adresse nach dem zuletzt zugewiesenen Speicherbereich. Wurde zuletzt ein Speicherbereich der Größe k an Adresse p vergeben, wird die Suche an der Adresse $p + k$ begonnen, sofern diese Adresse gültig ist. Anderenfalls wird am Anfang des Speichers begonnen. Hierbei ist der Fall zu berücksichtigen, dass wenn die Suche am Ende des Speichers angekommen ist, es unter Umständen Bereiche am Anfang gibt, die noch nicht untersucht wurden. Abbildung 4.9 zeigt exemplarisch die Speicherbelegung nach Anwendung dieser Strategie. Bitte beachten Sie, dass die zwischengespeicherte Adresse des zuletzt vergebenen Speicherbereichs nur dann verändert werden soll, wenn ein Speicherbereich mit Next-Fit alloziert wurde.

Abbildung 4.9: Allokation von zwei Byte mit Next-Fit

Best-Fit

Die Allokationsstrategie *Best-Fit* liefert den kleinsten passenden Speicherbereich zurück. Dazu muss für jede Anfrage der gesamte Speicherbereich, beginnend bei der ersten Adresse, durchsucht werden. Wird ein exakt passender Speicherbereich gefunden, kann die Suche vorzeitig abgebrochen und der gefundene Speicherbereich als Ergebnis zurückgegeben werden. Abbildung 4.10 zeigt exemplarisch die Speicherbelegung nach Anwendung dieser Strategie.

Abbildung 4.10: Allokation von zwei Byte mit Best-Fit

Worst-Fit

Die Allokationsstrategie *Worst-Fit* liefert im Gegensatz zu *Best-Fit* den größten passenden Speicherbereich zurück. Auch hier wird mit der ersten Adresse begonnen. Die Suche nach einem solchen Speicherbereich kann nur dann vorzeitig abgebrochen werden, falls ein freier Speicherbereich gefunden wurde, dessen Größe mindestens die Hälfte des insgesamt vorhandenen dynamischen Speichers beträgt. Abbildung 4.11 zeigt exemplarisch die Speicherbelegung nach Anwendung dieser Strategie.

Abbildung 4.11: Allokation von zwei Byte mit Worst-Fit

Lernerfolgsfragen

- Kann die *Next-Fit* Strategie mit Hilfe der *First-Fit* Strategie implementiert werden? Wie oft würde der Speicher im schlechtesten Fall nach einer freien Speicherstelle durchsucht werden?

4.4 Optimierte Speicherbereinigung

In Versuch 3 wurde die automatische Speicherbereinigung bei Beendigung eines Prozesses eingeführt. Durch den neu hinzugekommen externen SRAM ist ein vollständiges Durchsuchen des nun deutlich größeren Speichers dabei nicht länger praktikabel. Daher ist es notwendig, die automatische Speicherbereinigung beim Beenden eines Prozesses durch Einschränkung des Suchbereichs zu optimieren. Im Fall von Prozessen, welche Speicher auf keinem oder nur dem internen Heap allozieren, muss der externe Heap beispielsweise nicht durchsucht werden. Kurzlebige Prozesse, welche überhaupt keinen Speicher allozieren, würden bei vollständiger Speicherdurchsuchung deutliche Performanz Probleme hervorrufen.
Wir stellen im Weiteren einen Ansatz zur Optimierung vor, an welchem Sie sich orientieren können. Auch die exakte Umsetzung im Code bleibt Ihnen überlassen.

Allokationsrahmen Wir beschränken den zu durchsuchenden Adressraum auf einen reduzierten Teilabschnitt, welchen wir bei Heapinteraktionen (`malloc, free`, etc.) anpassen.
Der zu durchsuchende Teilabschnitt kann durch zwei Variablen `allocFrameStart` und `allocFrameEnd` markiert werden. Alle von einem Prozess allozierten Bereiche befinden

sich stets zwischen diesen beiden Adressen. Wenn Speicher neu alloziert oder freigegeben wird, verschieben sich diese Grenzen gegebenenfalls. Abbildung 4.12 veranschaulicht das Prinzip. Finden Sie einen geeigneten Speicherort für die Variablen. Machen Sie sich Gedanken, mit welcher bereits verwendeten Datenstruktur diese geschickterweise zu assozieren sind.

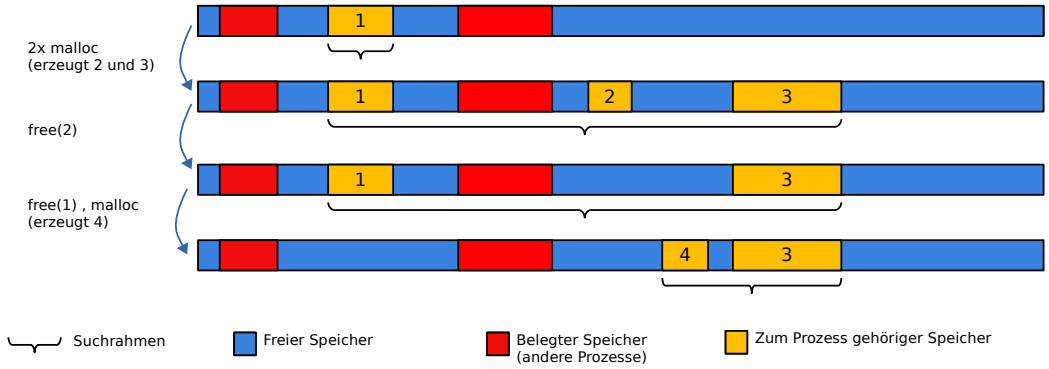

Abbildung 4.12: Allokationsrahmen mit verschiedenen Heapinteraktionen

4.5 Hausaufgaben

Achtung

Passen Sie das `define VERSUCH` auf die aktuelle Versuchsnummer an.

Implementieren Sie die in den nächsten Abschnitten beschriebenen Funktionalitäten. Halten Sie sich an die hier verwendeten Namen und Bezeichnungen für Variablen, Funktionen und Definitionen.
Lösen Sie alle hier vorgestellten Aufgaben zu Hause mithilfe von Atmel Studio 7 und schicken Sie die dabei erstellte und funktionsfähige Implementierung, einschließlich des Atmel Studio 7-Projekts, über die Praktikumswebsite ein. Beachten Sie bei der Bearbeitung der Aufgaben die angegebenen Hinweise zur Implementierung! Ihr Code muss ohne Fehler und ohne Warnungen kompilieren.

Achtung

Verwenden Sie zur Prüfung auf Warnungen den Befehl „Rebuild Solution“ im „Build“-Menü des Atmel Studio 7. Die übrigen in der grafischen Oberfläche angezeigten Buttons führen nur ein inkrementelles Kompilieren aus, d.h. es werden nur geänderte Dateien neu kompiliert. Warnungen und Fehlermeldungen in unveränderten Dateien werden dabei nicht ausgegeben.

4.5.1 SPI

Zur Verwendung des integrierten SPI-Moduls sind drei Basisfunktionen vorgesehen: `os_spi_init`, `os_spi_send` und `os_spi_receive`. Legen Sie die Dateien `os_spi.c/.h` an und implementieren Sie die folgende Funktionalität dort entsprechend der in Abschnitt 4.3.1 beschriebenen Grundlagen.

Initialisierung `os_spi_init` soll zunächst die Konfiguration der für den Bus relevanten I/O-Pins vornehmen und anschließend den Mikrocontroller als Master des SPI-Busses aktivieren. Wählen sie den mit dem 23LC1024 verträglichen SPI-Modus sowie eine geeignete Taktfrequenz zur Kommunikation.

Achtung

Wählen Sie die schnellste Taktfrequenz, die im Zusammenhang mit dem 23LC1024 unterstützt wird. Eine unnötig niedrige Busgeschwindigkeit gilt nicht als korrekte Lösung.

Unsere Implementierung verwendet nicht die SPI-Interrupt Routine. Das bedeutet das `SPIE`-Bit darf nicht aktiviert werden.

Senden und Empfangen von Bytes Implementieren Sie `os_spi_send` und `os_spi_receive` zum Senden und Empfangen einzelner Bytes über das SPI-Modul. Die Datenübertragung darf in keinem Fall unterbrochen oder gestört werden und muss daher entsprechend geschützt werden. Auf den Abschluss der Übertragung soll hier *aktiv gewartet* werden.
Da eine Übertragung immer zweiseitig ist, muss beim Empfangen von Daten ein Dummy-Byte gesendet werden. Verwenden Sie `0xFF` als Dummy-Byte und machen Sie sich die Ähnlichkeit des Sende- und Empfangsvorgangs bei der Implementierung zunutze. Wählen Sie dazu einen geschickten Rückgabewert bei `os_spi_send`. Beim Sendevorgang werden die empfangenen Daten verworfen und beim Empfangsvorgang sind die gesendeten Daten bedeutungslos.

4.5.2 23LC1024

Der externe SRAM wird analog zum internen SRAM wie in Versuch 3 durch drei Funktionen angesteuert: Initialisierung, Schreiben und Lesen.
Um die Kommunikation mit dem 23LC1024 übersichtlich und leserlich zu gestalten empfiehlt es sich für konstante Werte (zum Beispiel Bitkodierung von Befehlen) geeignete von defines anzulegen. Sich wiederholende Programmabschnitte lagern Sie bitte in eigene Unterfunktionen aus.

Größenreduktion Der gesamte externe SRAM bietet eine Speicherkapazität von 128 KiB, welche einem 17-bit Adressraum entsprechen. Da wir in Versuch 3 bereits den `MemAddr` Datentyp auf 16-bit festgelegt haben und der nächst größere Datentyp (32-bit) deutlich überdimensioniert ist, beschränken wir unsere Implementierung auf 16-bit. Das bedeutet, dass in SPOS nur 64 KiB des externen Speichers genutzt werden können. Verwenden Sie dabei die untere Hälfte des Adressraums im 23LC1024. Beginnen Sie also bei Adresse 0.

Initialisierung Zunächst muss der zuvor implementierte Treiber für das SPI-Modul aktiviert werden und ggf. weitere Steuerleitungen konfiguriert werden. Stellen Sie danach sicher, dass der externe SRAM nicht als SPI-Slave selektiert ist.

Stellen Sie hier auch den byteweisen Zugriffsmodus für den 23LC1024 durch Übertragen des entsprechenden Befehles ein.

Lesen und Schreiben Beide Methoden haben einen ähnlichen Aufbau. Wählen Sie zuerst den SPI-Slave durch manuelles Beschalten des CS-Pins aus. Nun ist dieser für die Buskommunikation aktiviert und Sie können die Übertragung mit Hilfe der SPI-Register beginnen. Die konkreten Inhalte der Übertragung entnehmen Sie der Dokumentation im Datenblatt des 23LC1024. Deaktivieren Sie zuletzt wieder den SPI-Slave und schützen Sie die Funktion im Kontext des Betriebssystems.

4.5.3 Treiber und Heap für den externen SRAM

Legen Sie eine Treiberinstanz `extSRAM` vom Typ `MemDriver` in `os_mem_drivers.c/.h` an. Diese muss zum einen Zeiger auf die in Abschnitt 4.5.2 implementierten Zugriffsfunktionen und zum anderen die für den externen SRAM geltenden Kenngrößen enthalten. Analog zur Heap-Datenstruktur für den internen SRAM soll nun selbige für den externen SRAM erstellt werden. Legen Sie die Datenstruktur in der `os_memheap_drivers.c/.h` an. Diese soll alle charakteristischen Werte des externen Heaps und einen Zeiger auf die Treiberinstanz des externen SRAMs enthalten. Initialisieren Sie den Heap des externen SRAM in geeigneter Form. Beachten Sie, dass der externe SRAM zwar eine Größe von 64 KiB hat, die durch den Heap genutzte externe Speicherkapazität jedoch ein Vielfaches von drei sein muss.

Lernerfolgsfragen

- Welche Größen definieren in SPOS ein Speichermedium?
  - Gibt es Redundanzen? Begründen Sie Ihre Antwort.
  - Was sind die Vor- und Nachteile, (keine) Redundanzen in den charakteristischen Attributen des Datenträgertreibers zu speichern?
- Erwarten Sie Probleme, wenn Sie den Heap/die Map bei der Initialisierung nicht vollständig mit Nullen überschreiben?
- Wie viel Byte des externen SRAM können insgesamt genutzt werden? Wie groß ist der SRAM insgesamt?
- Woraus ergibt sich die Tatsache, dass die nutzbare externe Speicherkapazität ein Vielfaches von drei Byte ist?

4.5.4 Anpassen der bisherigen Implementierung

Neben der Implementierung neuer Funktionalität, wie die Unterstützung eines externen Speichers, soll auch schon bestehende Funktionalität angepasst und erweitert werden.

Taskmanager

In den vorherigen Versuchen wurden einige Funktionen implementiert, die vom Taskmanager genutzt werden und nun einer Überarbeitung bedürfen. Da es in der Entwicklungsumgebung AVR Studio nicht möglich ist, sich den Inhalt des externen SRAM anzusehen (im Gegensatz zum internen SRAM), ist diese Funktionalität im Taskmanager integriert. Damit sie genutzt werden kann, müssen die Funktionen `os_getHeapListLength()` und `os_lookupHeap(uint8_t index)` entsprechend angepasst werden. Letztere muss nun `extHeap` zurückgeben, wenn der Parameter `1` übergeben wurde.
Bitte beachten Sie, dass der Taskmanager die von Ihnen implementierten Treiberfunktionen nutzt, um den Inhalt der Datenspeicher anzuzeigen. Entsprechend erfolgt eine korrekte Anzeige nur dann, wenn Ihre Implementierung korrekt ist. Dennoch kann diese Funktionalität des Taskmanagers das Debugging Ihres Codes erleichtern.

4.5.5 Reallokation

Implementieren Sie die Funktion `os_realloc`, welche die Größe eines Speicherbereiches dynamisch verkleinern bzw. vergrößern kann und dabei die in Abschnitt 4.3.6 aufgezeigten Probleme löst. Überlegen Sie sich, welche Randfälle auftreten können und welche Möglichkeiten existieren, um einen Speicherbereich zu vergrößern. Ein Speicherbereich soll nur dann an eine neu allozierte Speicherstelle verschoben werden, falls der Bereich nicht mit den freien Speicherzellen direkt vor sowie direkt nach seinem Bereich erweitert werden kann, um die gewünschte Speichergröße zu erreichen. Die Erweiterung in Speicherzellen direkt nach dem derzeit allozierten Bereich ist stets vorzuziehen. Ist es jedoch notwendig in Speicherzellen vor dem derzeit allozierten Bereich zu expandieren, so soll der gesamte Bereich so weit wie möglich nach vorne verschoben werden, um den Speicher zu defragmentieren. Erst wenn auch dies nicht möglich ist, soll nach einem geeigneten Bereich im restlichen Heap gesucht werden. Die Funktion hat folgende Signatur:

```
MemAddr os_realloc(Heap* heap, MemAddr addr, uint16_t size)
```

Falls es nicht möglich ist, den durch `addr` angegebenen Speicherbereich zu vergrößern bzw. zu verschieben, also nicht genügend Speicher zur Verfügung steht, muss der übergebene Speicherbereich unverändert bleiben und 0 zurückgeliefert werden. Beachten Sie auch, dass ein Prozess ausschließlich von ihm selbst allozierten Speicher reallozieren darf.

Lernerfolgsfragen

- Was kodiert der Rückgabewert von `os_realloc` im Normallfall, wenn die Größenanpassung erfolgreich war?

4.5.6 Heap Cleanup und Optimierung

Implementieren Sie, falls Sie dies im vorherigen Versuch noch nicht getan haben, die Funktion `void os_freeProcessMemory(Heap* heap, ProcessID pid)`, welche den gesamten Speicher freigibt, der von dem Prozess mit der angegebenen Prozess-ID auf dem angegebenen Heap alloziert wurde.
Fügen Sie anschließend in der Funktion `os_dispatcher` bzw. `os_kill` einen Aufruf dieser Funktion ein, um den Speicher des terminierten Prozesses sowohl im internen als auch im externen Heap automatisch freizugeben.
Entwickeln Sie zuletzt eine Optimierungsstrategie damit nicht immer der gesamte Speicher durchsucht werden muss.

4.5.7 Allokationsstrategien

Implementieren Sie alle in Abschnitt 4.3.7 vorgestellten Verfahren zur Ermittlung eines passenden Speicherbereiches, falls Sie diese nicht schon im vorherigen Versuch implementiert haben. Dabei soll es möglich sein, für jedes Speichermedium eine eigene unabhängige Allokationsstrategie auszuwählen. Berücksichtigen Sie dies bei der Implementierung.

Spezifische Hinweise zur Implementierung, um ungewollte Seiteneffekte mit den Testprogrammen zu vermeiden:

- Nextfit:
  - Das Aktualisieren der Adresse des zuletzt allozierten Speicherbereichs in anderen Strategien als Nextfit kann zum Fehlerfall im Testtask (4 Alloc Strategies) führen.

4.5.8 Zusammenfassung

Folgende Übersicht listet alle Typen, Funktionen und Aufgaben auf. Alle aufgelisteten Punkte müssen zur Teilnahme am Versuch bis zur Abgabefrist bearbeitet und hochgeladen werden. Diese Übersicht kann als Checkliste verwendet werden und ist daher mit Checkboxen versehen.

- ☐ `os_spi:`
  - ☐ Basisfunktionen zum SPI-Modul:
    - ☐ `void os_spi_init(void);`
    - ☐ `uint8_t os_spi_send(uint8_t data);`
    - ☐ `uint8_t os_spi_receive();`
- ☐ `os_mem_drivers:`
  - ☐ Globale Variablen
    - ☐ `MemDriver extSRAM__` (sowie das `define extSRAM`)
  - ☐ Treiber für den externen SRAM:
    - ☐ Initialisierungsfunktion
    - ☐ Funktion zum Lesen
    - ☐ Funktion zum Schreiben
- ☐ `os_memheap_drivers:`
  - ☐ Globale Variablen
    - ☐ `Heap extHeap__` (sowie das `define extHeap`)
  - ☐ Heap für den externen SRAM:
    - ☐ Anpassung von `os_getHeapListLength`
    - ☐ Anpassung von `os_lookupHeap`
    - ☐ Anpassung von `os_initHeaps`
- ☐ `os_memory:`
  - ☐ `MemAddr os_realloc(Heap* heap, MemAddr addr, uint16_t size)`
  - ☐ `void os_freeProcessMemory(Heap* heap, ProcessID pid)`
  - ☐ Optimierung der Speicherbereinigung
- ☐ `os_memory_strategies:`
  - ☐ Allokationsstrategien
    - ☐ *Next-Fit*
    - ☐ *Best-Fit*
    - ☐ *Worst-Fit*

4.6 Präsenzaufgaben

Die folgenden Aufgaben müssen bis zum Ende des Praktikumstermins umgesetzt werden. Es empfiehlt sich, die Aufgaben schon vor dem Praktikumstermin zu erarbeiten und nach Möglichkeit umzusetzen. Es ist erlaubt, die Praktikumstermine nach Abschluss aller Präsenzaufgaben früher zu verlassen.

4.6.1 Testprogramme

Sie finden im Moodle eine Sammlung von Testprogrammen, mit denen die Funktionalität Ihres Betriebssystems zum aktuellen Entwicklungsstand getestet werden kann. Am Ende des Versuchs müssen alle nicht optionalen Testprogramme fehlerfrei laufen. Der korrekte Ablauf der Testprogramme gemäß der ebenfalls im Moodle verfügbaren Beschreibung wird während des Versuchs geprüft. Die Implementierungshinweise, Achtung-Boxen und Lernerfolgsfragen in diesen Unterlagen weisen meist auf notwendige Kriterien für den erfolgreichen Durchlauf der Testprogramme hin.
Wenn Ihre selbst entwickelten Anwendungsprogramme fehlerfrei unterstützt werden, starten Sie die Testprogramme. Wenn es mit diesen Probleme gibt, haben Sie wahrscheinlich bestimmte Anforderungen nicht erfüllt, oder gewisse Sonderfälle nicht bedacht. Diese Sonderfälle können bei der Erweiterung Ihrer Implementierung des Betriebssystems für spätere Versuche zu Folgefehlern führen. Ergänzen oder korrigieren Sie Ihr Projekt, wenn Probleme mit den Testprogrammen auftreten.

Achtung

Sollten Sie alle zur Verfügung gestellten Testprogramme erfolgreich ausführen können, ist dies keine Sicherheit dafür, dass Ihr Code fehlerfrei ist. Diese Testprogramme decken lediglich einen Teil der möglichen Fehler ab.

Hinweis

Sollten Ihr externer Speichertreiber noch nicht funktionieren, empfiehlt es sich die korrekte Funktionalität des SPI Moduls durch eigene Tests sicherzustellen. Beispielsweise können Sie das korrekte Setzen der Konfigurations-Register auf dem 23LC1024 überprüfen oder einzelne Speicherzellen ansprechen.

4.7 Pinbelegungen

Port	Pin	Belegung
Port A	A0	LCD Pin 1 (D4)
	A1	LCD Pin 2 (D5)
	A2	LCD Pin 3 (D6)
	A3	LCD Pin 4 (D7)
	A4	LCD Pin 5 (RS)
	A5	LCD Pin 6 (EN)
	A6	LCD Pin 7 (RW)
	A7	frei
Port B	B0	frei
	B1	frei
	B2	frei
	B3	SIO2*
	B4	\CS
	B5	SI
	B6	SO
	B7	SCK
Port C	C0	Button 1: Enter
	C1	Button 2: Down
	C2	Reserviert für JTAG
	C3	Reserviert für JTAG
	C4	Reserviert für JTAG
	C5	Reserviert für JTAG
	C6	Button 3: Up
	C7	Button 4: ESC
Port D	D0	frei
	D1	frei
	D2	frei
	D3	frei
	D4	frei
	D5	frei
	D6	frei
	D7	frei

Pinbelegung für Versuch 4 (*Externer Speicher*).
* Nicht für den Versuch relevant.

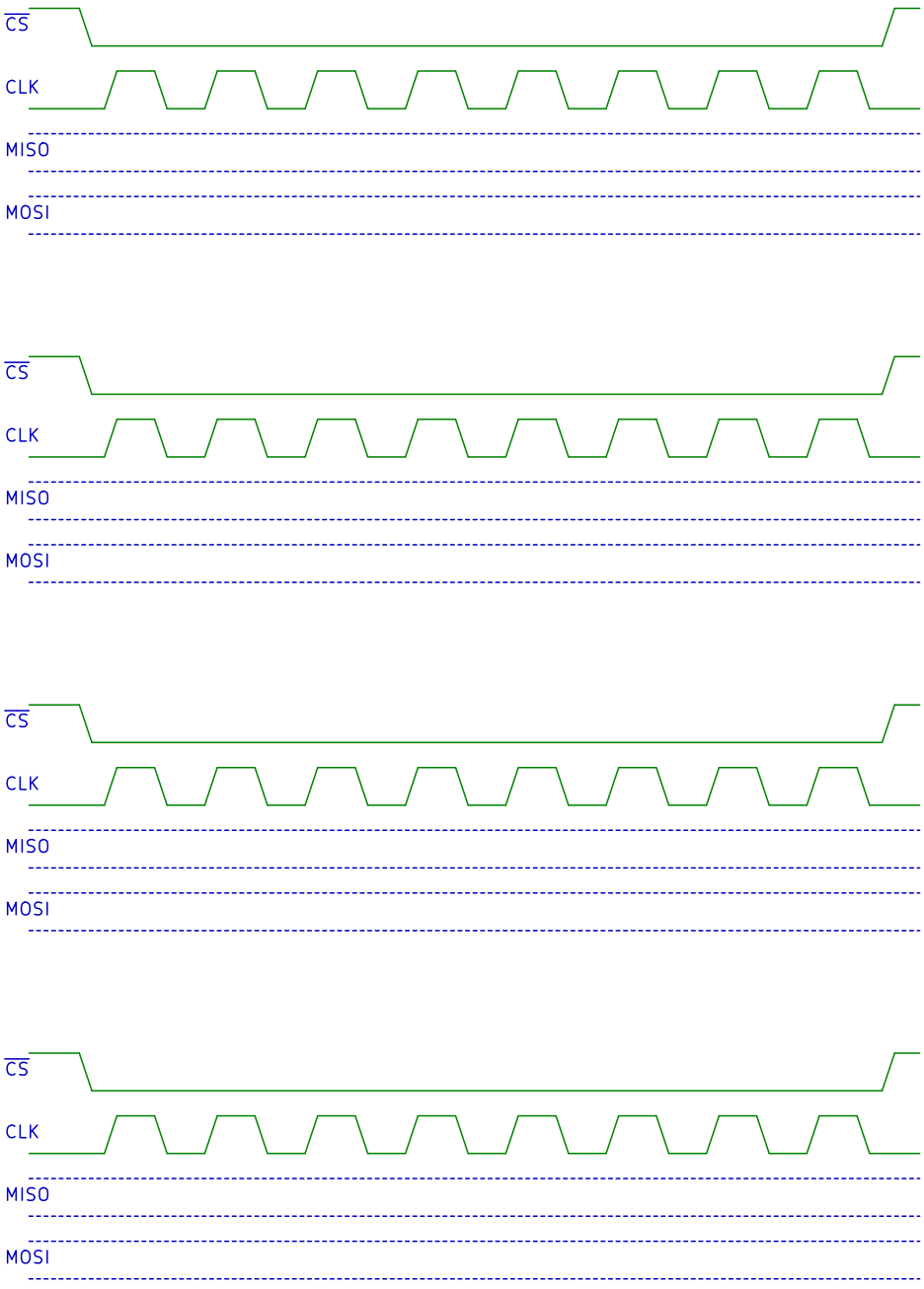

Abbildung 4.13: Leere SPI-Diagramme