Informatik 11
Embedded Software

RWTH AACHEN UNIVERSITY

Praktikum Systemprogrammierung

Versuch 6

RFID

Lehrstuhl Informatik 11 - RWTH Aachen

8. Dezember 2023

Commit: 48145a2f

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

6 RFID

In einem modernen Betriebssystem werden zur Kommunikation zwischen Anwendungsprogramm und Hardware Treiber eingesetzt. Treiber stellen Anwendungsprogrammen standardisierte Schnittstellen zur Verfügung. Somit bieten diese eine einfache Möglichkeit für Programme, auf die Hardware des Rechners zuzugreifen.

6.1 Versuchsinhalte

In den vergangenen Versuchen wurde das Betriebssystem SPOS implementiert. Dieses unterstützt bisher keine Methoden zur Benutzeridentifikation oder zur Verwaltung verschiedener Zugriffsrechte für den Taskmanager. In diesem Versuch wird ein in Teilen vorgegebener Treiber für eine RFID-Tag Leseplatine zu der bestehenden Implementierung von SPOS hinzugefügt und vervollständigt. Außerdem wird eine Benutzeridentifikation mittels RFID-Tags entwickelt. Schließlich wird das User Privileges Modul des Taskmanagers so angepasst, dass der Zugriff auf einige Funktionalitäten nur bestimmten, authentifizierten Benutzern zur Verfügung steht.

6.2 Lernziel

Das Lernziel dieses Versuchs ist das Verständnis der folgenden Zusammenhänge:

- Datenformat und Codierung von Radio Frequency Identification (RFID)-Tags auf Hardware- und Datenebene
- Entwicklung eines Treibers und eines Anwendungsprogramms zum RFID-Empfang
- Verwaltung verschiedener Zugriffsrechte

6.3 Grundlagen

In diesem Versuch wird die Identifikation von Benutzern mittels RFID behandelt. Um SPOS zu ermöglichen, Daten mit Hilfe von RFID zu empfangen, muss ein entsprechender Treiber zur Ansteuerung der dazu benötigten Hardware entwickelt werden.
Das folgende Kapitel enthält alle grundlegenden Informationen, die zum Verständnis von RFID und einer darauf basierten Benutzeridentifikation benötigt werden. Der erste Teil dieses Grundlagenkapitels behandelt zunächst die RFID-Technologie im Allgemeinen. Im Anschluss daran wird erläutert, wie eine Datenübertragung mit einem RFID-Tag abläuft. Danach werden Details zur Implementierung eines RFID-Treibers für SPOS erläutert und zuletzt die Benutzeridentifikation mit Hilfe der RFID-Daten erklärt.

6.3.1 RFID

RFID ist eine Technologie, die eine kabellose und berührungsfreie Datenübertragung ermöglicht. Die zu übertragenden Daten sind auf so genannten *RFID-Tags* (im folgenden Tag genannt) gespeichert und werden zu einem *RFID-Lesegerät* (im folgenden Leser genannt) übertragen. Ein Vorteil der RFID-Technologie ist, dass ein Tag üblicherweise keine eigene Spannungsversorgung benötigt: Die zum Betrieb benötigte Energie bezieht der Tag mittels Induktion aus einem elektromagnetischen Feld, welches vom Leser ausgestrahlt wird. Das elektromagnetische Feld oszilliert in einer bestimmten Frequenz; der zu lesende Tag muss auf diese Frequenz kalibriert sein. Der in diesem Versuch verwendete Leser erzeugt ein elektromagnetisches Feld mit einer Frequenz von 125 kHz. Die hier verwendeten Tags senden ein Bit pro 64 Oszillationen des Feldes, was zu einer Datenrate von etwa $2\,{}^{\mathrm{kBit}}/_{\mathrm{s}}$ führt.
Die Übertragung der Daten geschieht durch Bilden einer Signalfolge. Der Tag belastet das oben erwähnte elektromagnetische Feld des Lesers entsprechend dieser Signalfolge abwechselnd stärker und schwächer. Durch Messen der unterschiedlich starken Belastung dieses Feldes können die übertragenen Informationen erkannt werden.

6.3.2 Ablauf der Datenübertragung mittels RFID

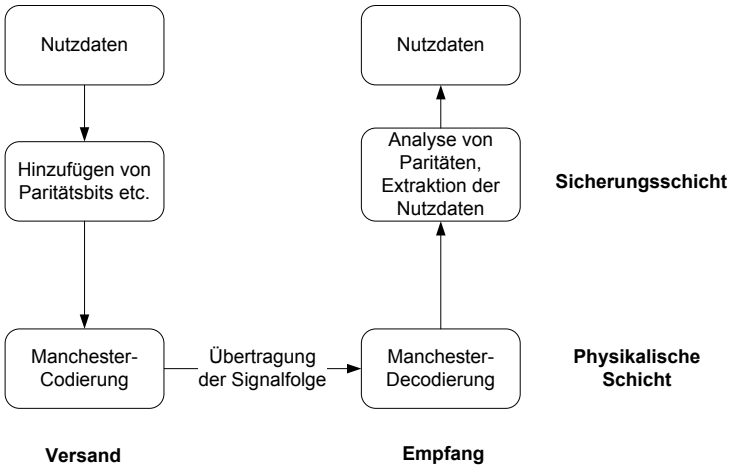

Abbildung 6.1: Ablauf der Datenübertragung mittels RFID

Die Nutzdaten, die mittels RFID übertragen werden sollen, werden für den Versand in zwei Schichten codiert, welche in Abbildung 6.1 dargestellt sind. In der *Sicherungsschicht* werden die zu übertragenen Daten unter anderem mit Paritätsinformationen versehen, um nach dem Empfang Fehler in der Übertragung zu erkennen. In der *physikalischen Schicht* werden diese Daten in Signalverläufe umgesetzt, die im *Manchester-Code* codiert sind. Diese Signalverläufe werden dann vom Tag zum Leser übertragen. Beim Empfang solcher Daten müssen die Codierungen der beiden Schichten wieder rückgängig gemacht werden. In den folgenden Abschnitten erhalten Sie Informationen, über die Datenüber-

tragung mittels RFID. Die Reihenfolge der Informationen orientiert sich an der Reihenfolge, in der diese beim Empfang benötigt werden: Zunächst wird die physikalische Datenübertragung von einem Tag zum Leser erläutert, dann der Manchester-Code und dessen Decodierung. Den Abschluss dieses Abschnitts bildet eine Erklärung der zusätzlichen Daten, die in der Sicherungsschicht eingefügt werden.

Leser

Ein Leser besteht neben weiteren Bauteilen aus einem integrierten Schaltkreis (Controllerchip) und einer Antenne, welche zum Aufbau des elektromagnetischen Feldes genutzt wird. Der Controllerchip erzeugt das Feld und misst dessen Belastung, welche durch einen in der Nähe befindlichen Tag hervorgerufen wird. Diese Änderungen werden dann nach einer Aufarbeitung als elektrisches Signal an einen angeschlossenen Mikrocontroller weitergegeben, der die Daten dekodiert und interpretiert. Bild 6.2 zeigt die verwendete Leser-Platine und einige der verwendeten Tags.

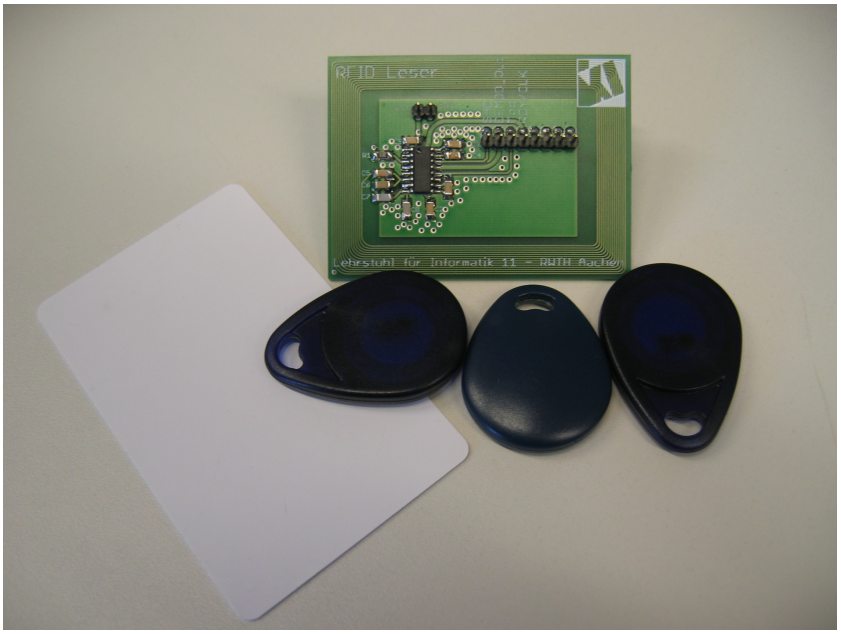

Abbildung 6.2: Der verwendete Leser (oben) und einige Tags (ein Tag im Scheckkartenformat und drei Tags im Schlüsselanhängerformat)

Der Leser wird auf PORT D aufgesteckt. In der Revision 1.1 des Lesers (abzulesen auf der Unterseite) müssen die Anschlüsse VCC und GND mit den entsprechenden Anschlüssen des Mikrocontrollerboards verbunden werden. Revision 1.2 verfügt über zusätzliche Steckverbindungen über die VCC und GND beim Aufstecken automatisch angelegt werden. Eine zusätzliche Verkabelung ist hier nicht nötig.

Manchester-Code

Die Daten, die ein Tag aussendet, sind im sogenannten Manchester-Code codiert. Dieser wurde entwickelt, um eine Folge von Datenbits zusammen mit einem Takt (Clock) in Flanken eines Signalverlaufs abzubilden. Es existieren zwei Definitionen für den Manchester-Code, die in diesem Versuch genutzte Hardware basiert auf der Definition nach G.E. Thomas. Der Manchester-Code nach Definition von Thomas wird vom Tag berechnet, indem die Funktion $\neg(\text{Datenbit} \oplus \text{Clock})$ gebildet wird. Abbildung 6.3 zeigt die Manchester-Codierung eines Beispielsignals nach der Definition von Thomas.

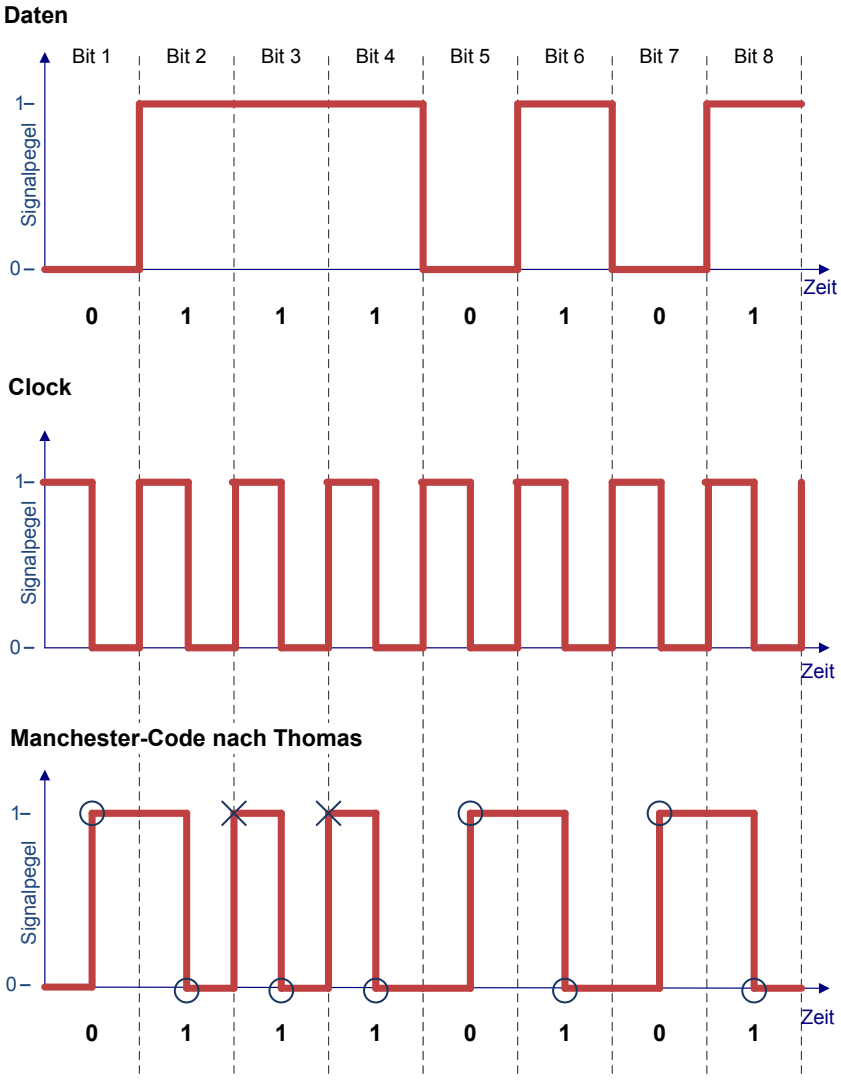

Abbildung 6.3: Manchester-Codierung eines Beispielsignals

Die Synchronisierung von Daten und Clock, in Verbindung mit der nach Thomas definierten Abbildung, stellt sicher, dass die Codierung einer 1 im Datensignal immer einer

fallenden und die Codierung einer `0` im Datensignal immer einer steigenden Flanke im Manchester-Code entspricht. Wenn zwei aufeinander folgende gleiche Bits übertragen werden, müssen demnach auch zwei gleiche Flanken (steigend oder fallend) übertragen werden. Da dies physikalisch nicht möglich ist, kommen Flanken im Signal hinzu, die keinem Bit entsprechen (Füllflanken). Diese Flanken sind in Abbildung 6.3 mit einem Kreuz markiert, während Flanken, die einem Datenbit entsprechen, mit einem Kreis markiert sind. Füllflanken können daran erkannt werden, dass sie nach einer nur halb so kurzen Zeitspanne wie üblich auftreten, also nicht synchron mit dem Taktsignal sind.

Decodierung des Manchester-Codes Zur Decodierung eines Manchester-codierten Signals werden die Flanken daraufhin analysiert, ob sie steigend oder fallend sind und ob sie zu den Flanken gehören, die einem Bit entsprechen.
Zur Erkennung ob eine Flanke ein Bit repräsentiert, kann man den Zeitabstand zwischen zwei aufeinander folgenden Flanken betrachten. Im Idealfall gibt es genau 2 Zeitabstände in denen Flanken aufeinander folgen können: Entweder liegt die vorhergehende Flanke eine ganze Clock-Periode zurück, oder lediglich eine halbe. Ist der Abstand zur vorhergehenden Flanke größer als eine halbe Clock-Periode, dann spricht man von einer *langen Flanke*, im anderen Fall von einer *kurzen Flanke*. Lange Flanken treten immer dann auf, wenn das zu ihnen gehörige Datenbit vom vorherigen Datenbit verschieden ist. Sie bilden stets genau ein Datenbit ab.
Wenn mehrere gleiche Bits hintereinander übertragen werden, folgen kurze Flanken im Manchester-codierten Signal aufeinander, die abwechselnd zu ignorieren und zu verarbeiten sind (Übergang zwischen Bit 3 und 4 aus Abb. 6.3). Sind also die aktuelle Flanke und die davor liegende Flanke kurz, muss beachtet werden, ob die vorherige Flanke ignoriert wurde. Ist dies der Fall, wird die aktuelle Flanke ausgewertet. Wurde die vorherige Flanke ausgewertet, ist die aktuelle Flanke zu ignorieren. Zu Beginn einer derartigen Folge von kurzen Flanken folgt eine erste kurze Flanke auf eine lange Flanke (Übergang von Bit 2 zu Bit 3 in Abb. 6.3). Diese ist zu ignorieren.

Datencodierung in der Sicherungsschicht

Die Nutzdaten, die ein Tag überträgt, werden *Tag-ID* (oder kurz ID) genannt. Die ID eines Tags ist 40 Bit breit und wird nibbleweise (1 Nibble = 1 Halbbyte = 4 Bit) übertragen. Diese Datenbits der Nutzdaten werden während der Codierung in der Sicherungsschicht um weitere Daten ergänzt, um Paritätsprüfungen zu ermöglichen. Außerdem werden sie um einen Header ergänzt, der es ermöglicht, den Beginn einer Datenübertragung zu identifizieren. Als letztes wird ein Stoppbit angehängt. Insgesamt werden also für die Übertragung einer ID 64 Bit verwendet, deren jeweilige Funktion in Tabelle 6.1 dargestellt ist.
Die ersten 9 Bit stellen den Header dar. Sie haben immer den Wert `1`. Es folgen zehn 5 Bit große Datenpakete, wobei jeweils die ersten 4 Bit Nutzdaten der ID darstellen. Das 5. Bit jedes Datenpakets ist ein gerades Paritätsbit. Nach den Datenpaketen wird die Übertragung durch 4 Bit für Spaltenparitäten und ein Stoppbit abgeschlossen.

Bits	Funktion	Werte
$b_0 \dots b_8$	Header	alle '1'
$b_9 \dots b_{13}$	Datenpaket 1	$b_{13} = b_9 \oplus b_{10} \oplus b_{11} \oplus b_{12}$
$b_{14} \dots b_{18}$	Datenpaket 2	$b_{18} = b_{14} \oplus b_{15} \oplus b_{16} \oplus b_{17}$
$\vdots$	$\vdots$	$\vdots$
$b_{54} \dots b_{58}$	Datenpaket 10	$b_{58} = b_{54} \oplus b_{55} \oplus b_{56} \oplus b_{57}$
b_{59}	Spaltenparität 1	$b_{59} = b_9 \oplus b_{14} \oplus \dots \oplus b_{54}$
b_{60}, b_{61}, b_{62}	Spaltenparitäten $2, 3, 4$	Analog zu Spaltenparität 1
b_{63}	Stoppbit	'0'

Tabelle 6.1: Übersicht über die von einem Tag gesendeten Bits. Zu beachten ist die Indizierung der einzelnen Bits: b_0 bezeichnet hier das erste übertragene Bit, dieses stellt jedoch das Most-Significant-Bit der übertragenen Daten dar

Die Spaltenparitäten werden über die ersten, zweiten, dritten, bzw. vierten Datenbits der Pakete gebildet; über die Paritätsbits der Datenpakete wird keine Spaltenparität gebildet. Auch hier wird eine gerade Parität genutzt. Das Stoppbit hat immer den Wert 0. Ein Tag mit der ID 0x0123456789 wird also entsprechend der Tabelle 6.2 übertragen.

Lernerfolgsfragen

- Was ist die Aufgabe der physikalischen Schicht? Was ist die Aufgabe der Sicherungsschicht?
- Welche Codierung wird in der physikalischen Schicht verwendet? Welche in der Sicherungsschicht?
- Wie können bei der Decodierung von Manchester-Code Flanken gefunden werden, die keinem Bit entsprechen?
- Welchem Datenbit entspricht eine steigende (fallende) Flanke im Manchester-Code nach Thomas?
- Warum besteht der Header in der Sicherungsschicht aus genau neun 1-Bits? Können in den von einem Tag übertragenen Daten außerhalb des Headers neun aufeinanderfolgende 1-Bits auftreten?

Bits	Funktion
1111 1111 1	Header
0000 0	Codierung der '0' + Paritätsbit *(Datenpaket 1)*
0001 1	Codierung der '1' + Paritätsbit *(Datenpaket 2)*
0010 1	Codierung der '2' + Paritätsbit *(Datenpaket 3)*
0011 0	Codierung der '3' + Paritätsbit *(Datenpaket 4)*
0100 1	Codierung der '4' + Paritätsbit *(Datenpaket 5)*
0101 0	Codierung der '5' + Paritätsbit *(Datenpaket 6)*
0110 0	Codierung der '6' + Paritätsbit *(Datenpaket 7)*
0111 1	Codierung der '7' + Paritätsbit *(Datenpaket 8)*
1000 1	Codierung der '8' + Paritätsbit *(Datenpaket 9)*
1001 0	Codierung der '9' + Paritätsbit *(Datenpaket 10)*
0001 0	Spaltenparitäten + Stoppbit

Tabelle 6.2: Beispiel: Übertragung der ID 0x0123456789

6.3.3 Treiber

Bestandteile des Treibers

Der Empfang eines Tags geschieht in mehreren Schritten, die von verschiedenen Komponenten des Treibers übernommen werden. Der erste Schritt des ID-Empfangs ist der Empfang des Signals am Leser. Jede Flanke in diesem Eingangssignal wird auf ihren Wert (steigend/fallend) und ihren Abstand von der vorigen Flanke hin analysiert. Flanken, die auf Grund eines zu kurzen oder zu langen Abstandes von der vorigen Flanke als Signalrauschen identifiziert werden konnten, werden aussortiert. Den Empfang und die Analyse des Eingangssignals übernehmen die bereits vorgegebenen Interrupt Service Routinen (ISR) `TIMER1_CAPT_vect` und `TIMER1_OVF_vect`, welche die Abstände und Werte der empfangenen Signalflanken speichern. Die ISRs werden im nächsten Abschnitt genauer erläutert.
Die Funktion `rfid_receive` analysiert im zweiten Schritt die empfangenen Flanken und gibt die decodierte ID zurück. Die Funktion koordiniert die Decodierung der physikalischen Schicht und der Sicherungsschicht. Es wird ein Unterprogramm zur Decodierung des Manchester-Codes aufgerufen. Anschließend wird die so erhaltene Bitfolge durch weitere Unterfunktionen auf einen korrekten Header und korrekte Paritäten überprüft. Wenn kein Fehler im aktuellen Tag vorliegt, kann die ID des Tags als dritter Schritt extrahiert und zurückgegeben werden. Im Folgenden werden die einzelnen Funktionalitäten und Konzepte zu deren Umsetzung näher erläutert und der genaue Ablauf der `rfid_receive` Funktion erklärt.

Abbildung 6.4 zeigt eine Übersicht über die Datenflüsse, die beim Empfang einer Tag-ID wichtig sind. Die darin enthaltenen Funktionen, Speicherbereiche und Datenflüsse werden zusammen mit der Funktionsweise des Datenempfangs im Folgenden erläutert.

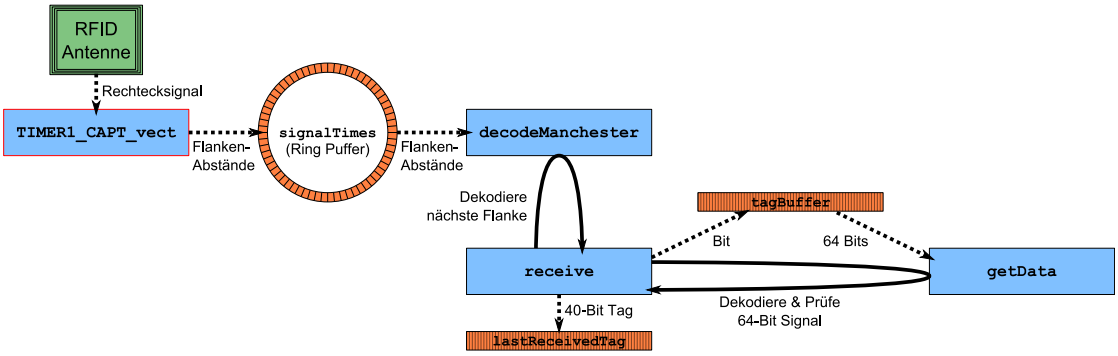

Abbildung 6.4: Übersicht der Datenflüsse beim Empfang von Daten durch den RFID-Treiber. Datenstrukturen sind orange dargestellt, Routinen hingegen blau. Hardware ist grün.

Empfang des Eingangssignals

Sämtliche Daten des RFID Moduls werden in der globalen Variablen `rfid` abgelegt. Diese enthält eine Instanz der Struktur `RFID`. Die beiden Interrupt Service Routinen `TIMER1_CAPT_vect` und `TIMER1_OVF_vect` sind bereits vorgegeben und empfangen die Daten des Lesers. Dazu messen sie den Abstand einer Signalflanke von der vorherigen. Sie speichern zu jeder Signalflanke den zeitlichen Abstand, den die Flanke von der vorhergehenden Flanke hat. Signalflanken, die aus Signalrauschen entstanden sind, werden ignoriert. Solche Flanken sind dadurch charakterisiert, dass sie einen sehr kurzen Abstand von der vorigen Flanke haben. Die ISRs verwenden zum Speichern der Flankenabstände den als Ringpuffer implementierten privaten Speicherbereich mit der `intSRAM` Adresse `rfid.signalTimes`. Da die Signalflanken immer abwechselnd steigend und fallend sind (auf eine steigende Flanke kann nur eine fallende Flanke folgen), wird nur die Richtung der ersten Flanke des Ringpuffers im Attribut `rfid.signalStartEdge` gespeichert. Alle weiteren Flankenrichtungen lassen sich abhängig von ihrem Index daraus errechnen. Zur Codierung von steigenden und fallenden Flanken werden die Konstanten `1` bzw. `0` verwendet. Zur Koordination des Schreibzugriffs auf den Puffer wird das Attribut `rfid.head` als Index auf den nächsten zu schreibenden Wert verwendet.

Reservierung von Speicher für die Daten-Arrays Verschiedene Funktionen führen Lese- und Schreibzugriffe auf den Ringpuffer `rfid.signalTimes` durch. Da diese Funktionen alle von nur einem Anwendungsprogramm genutzt werden (also durch die Funktion `rfid_receive` aufgerufen werden) oder ISRs sind, wird dieser Speicherbereich als privater Speicher auf dem Heap alloziert. Diese Allokation geschieht in der Funktion `rfid_init`, die als erstes im entsprechenden Anwendungsprogramm aufgerufen werden

muss. Diese Funktion aktiviert die zuvor beschriebenen ISRs und stellt diesen die Speicheradresse des Ringpuffers zur Verfügung, um einen direkten Speicherzugriff und damit mögliche Abspeicherung der vom Leser gemessenen Signallängen zu ermöglichen.
Es ist nicht möglich, den Speicherbereich als gemeinsamen Speicher anzulegen. Die entsprechenden Zugriffsfunktionen weisen keine für die Verarbeitung der vom Tag ausgesendeten Daten ausreichende Geschwindigkeit auf und es könnte zu Datenverlust kommen.

Achtung

Nach dem Einbinden des RFID-Treibers müssen ggf. die Kenngrößen für den Heap aktualisiert werden, um auf den anwachsenden Block der globalen Variablen zu reagieren!

Funktion zum Empfang von Tag-IDs

Die Funktion `rfid_receive` hat die Aufgabe, die von der ISR empfangenen Rohdaten zu analysieren und weiter zu verarbeiten. Dazu verwendet sie die weiter unten beschriebenen Hilfsfunktionen `rfid_decodeManchester` und `rfid_getData`. `rfid_receive` greift lesend auf den Ringpuffer `rfid.signalTimes` und die Variable `rfid.signalStartEdge` zu. Der Index `rfid.tail` wird beim Zugriff genutzt, um die aktuelle Leseposition innerhalb des Ringpuffers zu speichern. Zum Speichern von bereits empfangenen Teilen der Tag-IDs verwendet `rfid_receive` einen Tag-Puffer. Diesen bildet das Attribut `rfid.tagBuffer`. Dieser Puffer ist 64 Bit groß, weshalb die empfangenen Daten bitweise und nicht wie bei einem Array byteweise geschrieben werden müssen. Hierbei bietet es sich an, den Puffer wie ein Schieberegister zu verwalten. Dabei wird der Puffer bei jedem Schreibzugriff um 1 Bit nach links geshiftet und das neu zu schreibende Bit an den Index 0 geschrieben. Nach Einlesen von 64 Bits liegen dann alle empfangenen Bits am richtigen Index.
Um eine ermittelte ID in anderen Anwendungsprogrammen, wie z. B. zur Benutzeridentifikation (siehe Kapitel 6.3.4) weiterverarbeiten zu können, soll sie im Attribut `rfid.lastReceivedTag` gespeichert werden. `rfid_receive` soll außerdem in der Lage sein, eine empfangene ID optional auch auf dem LCD ausgeben zu können. Dazu soll das `define DISPLAY_RFID_TAG_AFTER_RECEIVE` ausgewertet werden.

Manchester-Dekodierung Die bereits angelegte, jedoch noch zu ergänzende Funktion `rfid_decodeManchester` soll empfangene Signalpegel analysieren und erkennen, ob mit einer bestimmten Flanke ein Bit übertragen wurde. Wenn ein Bit übertragen wurde, soll sie analysieren, welchen Wert dieses Bit hatte. Die Funktion wird für jede neue Flanke aufgerufen. Als Parameter erhält diese Funktion Informationen über die Art der zu untersuchenden Flanke (`1` für steigende und `0` für fallende Flanken) und den zeitlichen Abstand der zu untersuchenden zur vorherigen Flanke.

Die Funktion `rfid_decodeManchester` untersucht diese Informationen, ob es sich bei der zu untersuchenden Flanke um ein Datenbit oder eine Füllflanke handelt. Wenn sie einem Bit entspricht, gibt die Funktion als Ergebnis das entsprechende Bit zurück; andernfalls soll ein Fehlerwert (> 1) zurückgegeben werden. Ein derartiger Fehlerwert markiert das Bit als nicht decodierbar. Bei der Analyse der Flanken ist der entsprechende Abschnitt zur Manchestercodierung aus Kapitel 6.3.2 im Grundlagenkapitel zu beachten. Da bereits in den ISRs mit Hilfe der Konstanten `NOISE_THRESHOLD` eine Rauschunterdrückung durchgeführt wird, ist an dieser Stelle eine Behandlung zu kurzer Flanken nicht notwendig. Zur Unterscheidung von kurzen, langen und zu langen Flanken bietet es sich an, die `defines TICKSRATIO` und `NOISE_THRESHOLD` zu verwenden.
`rfid_decodeManchester` wird für jede Flanke einzeln aufgerufen. Die Information, ob die letzte Flanke ignoriert wurde (d.h. ob die aktuelle Flanke ggf. ignoriert werden muss), muss zwischen mehreren Funktionsaufrufen erhalten bleiben. Dazu kann diese Information beispielsweise in einem Attribut, etwa `rfid.lastManchesterSkipped`, in der `RFID` Struktur gespeichert werden. Eine andere Möglichkeit ist die Verwendung einer lokalen Variablen, die mit dem Schlüsselwort `static` deklariert wurde. Weitere Informationen zum Schlüsselwort `static` sind im Begleitendes Dokument für das Praktikum Systemprogrammierung zu finden. Bedenken Sie hierbei auch, wie diese Information zu behandeln ist, wenn Sie eine lange Flanke lesen.

Header-Erkennung Ein Header wird folgendermaßen erkannt: Die neun Most-Significant-Bits des Tag-Buffers müssen den Wert `1` haben, da der Header aus neun `1`-Bits besteht. Ist dies nicht der Fall, liegt kein vollständiges Paket vor und sie müssen den Buffer nicht auf weitere Daten prüfen. Es bleibt Ihnen überlassen diese Funktionalität zur Prüfung des Header in eine extra Funktion auszulagern oder in der `rfid_receive` zu implementieren.

Paritätsprüfungen Die Funktion `rfid_getData` hat die Aufgabe, die Daten, die auf den Header eines Tags folgen, auf die in Tabelle 6.1 angegebenen Paritäten zu prüfen und ggf. die Nutzdaten aus dem Tag extrahieren und zurückgeben. Die Funktion ist bereits angelegt, aber noch nicht implementiert.
Diese Funktion soll die im Tag-Puffer abgelegten Daten hinsichtlich Spalten- sowie Zeilenparitäten überprüfen. Außerdem muss das Stopp-Bit geprüft werden. In welcher Reihenfolge Sie dies tun, ist Ihnen überlassen.
Wenn an einer Stelle der Prüfungen ein Fehler aufgetreten ist, soll der Wert `0` zurückgegeben werden, welcher keinen gültigen Tag darstellt.
Nachdem alle 64 Bit empfangen wurden und wenn dabei keine Fehler aufgetreten sind, können die Nutzdaten, also die ID aus dem Tag-Puffer extrahiert werden. Dann sollte der Tag-Puffer geleert werden, um das nächste Tag speichern zu können. Zuletzt wird die extrahierte ID zurückgegeben.

Weiterverarbeiten der ID Wenn eine ID empfangen wurde, kann `rfid_receive` sie weiterverarbeiten. Eine empfangene ID soll immer global abgespeichert werden; zu die-

sem Zweck ist das Attribut `rfid.lastReceivedTag` zu verwenden. Je nach Wert des `define DISPLAY_RFID_TAG_AFTER_RECEIVE` soll sie außerdem in Hex-Darstellung auf dem LCD ausgegeben werden.

Lernerfolgsfragen

- Aus welchen Funktionen besteht der RFID-Treiber?
- Welche(r) Teil(e) des RFID-Treibers übernimmt die Decodierung der Daten in der physikalischen Schicht?
- Welche(r) Teil(e) übernimmt die Decodierung der Daten in der Sicherungsschicht?
- Wieso muss der Speicherbereich `rfid.signalTimes` noch angelegt werden? Wie und wo werden diese Daten angelegt?
- Wie kann `rfid_receive` feststellen, ob die ISRs neue Werte empfangen haben?
- Wie groß muss der Tag Puffer mindestens sein?
- Wie kann aus dem Inhalt des Tag-Puffers die Tag-ID extrahiert werden?

6.3.4 Benutzeridentifikation und Zugriffsrechte

Bislang unterstützt SPOS keine Benutzeridentifikation und keine Zugriffsrechte - jeder Nutzer mit physischem Zugriff auf das Mikrocontrollerboard kann beliebige Aktionen im Taskmanager durchführen. Um dieses Problem zu lösen wird in diesem Versuch ein Benutzeridentifikationssystem mit Hilfe von RFID implementiert. Das Benutzeridentifikationssystem soll einen Benutzer anhand der ID eines Tags erkennen und einloggen, und je nach Benutzerklasse verschiedene Taskmanager-Funktionen freischalten. Zunächst wird hier das System der Benutzerverwaltung näher erläutert, dann folgt eine Beschreibung der verschiedenen Berechtigungsstufen.

Benutzerverwaltung

Im Modul `os_user_privileges` ist ein Interface vorgegeben über welches der Taskmanager mit einem Rechtesystem erweitert werden kann. In der aktuellen Version von SPOS werden zu jedem Zeitpunkt alle Rechte gewährt. Dies erkennt man daran, dass die Funktion `os_askPermission` stets `OS_AP_ALLOW` zurückliefert.
In der Datei `os_user_privileges.h` ist ein `enum PermissionRequest` vorgegeben, welches für jede Seite des Taskmanagers einen eigenen Eintrag besitzt. Wählt ein Benutzer beispielsweise die Seite zum Löschen eines Heaps aus, fragt der Taskmanager zunächst

an, ob dies für den aktuellen User zulässig ist, indem er `os_askPermission` mit dem Parameter `OS_PR_ERASE_HEAP` aufruft. Zusätzlich wird die ID des zu löschenden Heaps im zweiten Parameter übergeben. Der dritte Parameter zeigt die Semantik des zweiten Parameters an, in diesem Fall also, dass es sich um eine Heap ID handelt. Auf diese Weise kann sehr genau gesteuert werden, welche Funktionalitäten ein Benutzer aufrufen darf und welche nicht. Falls eine Funktionalität verboten wird, kann im letzten Argument ein Pointer auf einen Prog-String an den Taskmanager übergeben werden, der eine detaillierte Fehlermeldung enthält.
Konsultieren Sie die Dokumentation der Datei `os_user_privileges.h` für weitere Informationen.

Benutzeridentifikation

Die Benutzeridentifikation soll mit Hilfe der Funktion `os_loginUser` im `os_user_privileges` Modul implementiert werden.
Ein Anwendungsprogramm wertet dazu Informationen über empfangene IDs vom RFID-Treiber aus. Eine empfangene ID wird abgespeichert und mit einer Liste von Benutzern verglichen, welche die Zugriffsrechte (siehe unten) der bekannten Benutzer beinhaltet. Anschließend, muss die berechnete Benutzerklasse an `os_loginUser` übergeben werden.

Benutzerklassen und Zugriffsrechte im Taskmanager

Der Taskmanager soll zwischen drei verschiedenen Szenarien unterscheiden können:

- Es ist *kein Benutzer* angemeldet (`OS_UT_NONE`)
- Eine Person ist als *Gast* angemeldet (`OS_UT_RESTRICTED`)
- Eine Person ist als *Benutzer* angemeldet (`OS_UT_USER`)

Wenn keine Person angemeldet ist, soll der Taskmanager beim Öffnen eine entsprechende Meldung ausgeben. Danach soll er sich wieder beenden - das Öffnen des Taskmanagers ist nur als Gast oder als angemeldeter Benutzer erlaubt. Wenn ein Gast angemeldet ist, soll der Taskmanager keine Möglichkeit anbieten, etwas am System zu ändern. Dies umfasst das Ändern von Prozessprioritäten, der Scheduling- und der Allokationsstrategie, sowie das Starten, Beenden, Pausieren oder Fortsetzen von Prozessen, sowie das Löschen von Heaps. Es soll Gästen gestattet sein, sich die aktuellen Prozesse und ihre Prioritäten, sowie die aktuelle Scheduling- und Allokationsstrategie und Heaps anzusehen. Wenn ein Benutzer angemeldet ist, soll ihm der volle Zugriff auf den Taskmanager gewährt werden, mit einer Ausnahme: Auch ein Benutzer darf keine Manipulationen an den zur Benutzeridentifikation und Zugriffssteuerung relevanten Prozessen durchführen und auch keine weiteren Kopien derselben starten. Hierzu müssen Sie die vom Taskmanager übergebenen Parameter auswerten um zu entscheiden, ob eine Funktionalität wie etwa `OS_PR_KILL` zulässig ist.
Wenn ein Gast oder ein Benutzer versucht, eine Funktion aufzurufen, für die er nicht genügend Zugriffsrechte besitzt, soll eine entsprechende Meldung auf dem LCD ausgegeben werden.

Der Taskmanager soll beim Beenden die aktuell angemeldete Person ausloggen (d.h. die aktuelle Benutzerklasse auf *nicht angemeldet* setzen), damit weitere Zugriffe gesperrt werden. Sie können hierzu den Aufruf des Taskmanagers im Scheduler ergänzen.

Lernerfolgsfragen

- Welche Benutzerklassen gibt es? Welche Aktionen im Taskmanager sind diesen erlaubt?
- Ist es nötig, die ID der aktuell angemeldeten Person zu speichern?
- Warum darf ein Benutzer keine Manipulationen an den zur Identifikation und Zugriffssteuerung relevanten Prozessen ausführen?
- Warum ist es sinnvoll, die aktuell angemeldete Person beim Beenden des Taskmanagers auszuloggen?

6.4 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 Microchip Studio 7 und schicken Sie die dabei erstellte und funktionsfähige Implementierung über Moodle ein.

Achtung

Alle im Versuch geforderten Funktionalitäten müssen implementiert sein!
Ihre Abgabe soll dabei die `.atsln`-Datei, das Makefile sowie den Unterordner mit den `.c/.h`-Dateien inklusive der `.xml/.cproj`-Dateien enthalten. Es sollen keine Testtasks, PDFs oder Doxygen-Dateien abgegeben werden.

Beachten Sie bei der Bearbeitung der Aufgaben die angegebenen Hinweise zur Implementierung! Ihr Code muss ohne Fehler und ohne Warnungen kompilieren sowie die Testtasks mit aktivierten Compileroptimierungen bestehen. Wie Sie die Optimierungen einschalten ist dem begleitenden Dokument in Abschnitt *6.3.2 Probleme bei der Speicherüberwachung* zu entnehmen.

Achtung

Verwenden Sie zur Prüfung auf Warnungen den Befehl „Rebuild Solution“ im „Build“-Menü des Microchip 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.

Hinweis

Da die Ports B und D für andere Zwecke benötigt werden, muss in diesem Versuch von der Benutzung des externen SRAMs abgesehen werden. Somit ist es ratsam, den externen SRAM wie auch den dazugehörigen Heap aus der Liste Ihrer Speichermedien bzw. Heaps zu entfernen.

6.4.1 Einleitung

Als Hausaufgabe sollen die im vorangegangenen Kapitel erläuterten Funktionalitäten (Treiber und Zugriffsrechte) konkret in SPOS implementiert werden. Dazu wird im Folgenden eine genauere Beschreibung einer sinnvollen Implementierung gegeben.

6.4.2 Treiber

Integrieren Sie das Treibergerüst für den Leser in Ihre bestehende Implementierung von SPOS. Beachten Sie, dass nachdem Sie das Treibergerüst eingebunden haben, der Speicherverbrauch der globalen Variablen angestiegen ist und Sie ggf. das entsprechende `#define HEAP_OFFSET` anpassen müssen. Sorgen Sie außerdem dafür, dass die Funktion `rfid_init` in dem Anwendungsprogramm aufgerufen wird, welches die Funktion `rfid_receive` nutzt, um eine Tag-ID auszulesen.

Defines Vervollständigen Sie folgende `defines` zur Ermittlung der Zeitkonstanten des RFID-Signals. Die Vorlage hierfür finden Sie in der Datei `os_rfid.c` und in Listing 6.1.

```
1 //! Frequency of the RFID tag
2 #define F_RFID_OSC    ?
3 //! Ratio of RFID Tag Frequency to RFID data rate
4 #define RFID_SCALER   ?
5 //! Prescaler used for timer 1
6 #define TIMER_SCALER  ?
7 //! Number of timer ticks for a long edge
8 #define TICKSRATIO    ?
```

Listing 6.1: `defines` zur Ermittlung der Zeitkonstanten des RFID-Signals.

Die `defines F_RFID_OSC`, `RFID_SCALER` und `TIMER_SCALER` sollen durch die expliziten Zahlenwerte definiert werden; der Wert für `TICKSRATIO` kann aus den vorhandenen `defines` und dem `define F_CPU` berechnet werden. Den verwendeten Prescaler für Timer 1 können Sie in der Funktion `rfid_init` ablesen. Das `define RFID_SCALER` gibt an, wieviele Oszillationen des vom Leser erzeugten Feldes einem Bit in der Datenübertragung entsprechen. Konsultieren Sie im Zweifel das Datenblatt des ATmega 644 und das Grundlagenkapitel.

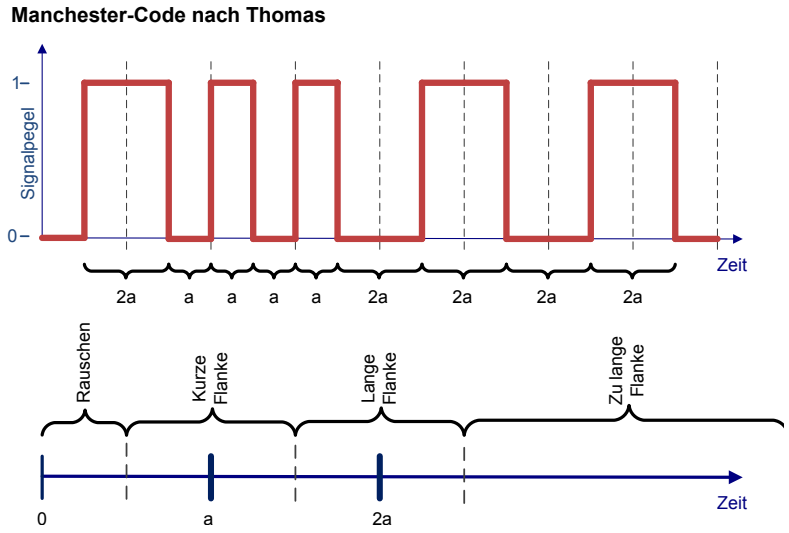

Abbildung 6.5: Veranschaulichung der für die Funktion `rfid_decodeManchester` benötigten Intervalleinteilung

Abbildung 6.5 zeigt im oberen Teil den Ausschnitt eines manchester-codierten Signals (vgl. Abbildung 6.3). Dieses Signal besteht aus kurzen Flanken (a) und langen Flanken ($2a$). Das dargestellte Signal ist in der Weise idealisiert, dass kein Rauschen im Signal vorhanden ist und Signallängen immer exakt eingehalten werden. Da das zu decodierende Signal jedoch diese Störgrößen beinhalten kann, muss die Funktion `rfid_decodeManchester` zwischen verschiedenen Fällen unterschieden. Diese sind im unteren Teil der Abbildung dargestellt. Die Zeitachse des oberen Teils der Abbildung ist im Vergleich zum unteren Teil gestaucht. Die zu errechnende Konstante `TICKSRATIO` entspricht in diesem Beispiel dem Wert $2a$. Als Intervallgrenzen zur Abgrenzung können die Werte $\frac{a}{2}$, $\frac{3a}{2}$ und $\frac{5a}{2}$ genutzt werden.

rfid_decodeManchester Implementieren Sie die Funktion `rfid_decodeManchester`. Die Methode analysiert eine Flanke anhand ihres Abstandes zur vorhergehenden Flanke darauf, ob sie ein Datenbit, eine Füllflanke oder eine Signalstörung darstellt. Hierzu können Sie die die im vorherigen Punkt definierten Konstanten benutzen. Wenn die Flanke einem Datenbit entspricht, muss die Methode den Wert des bit ermitteln. Hierzu können Sie die Information nutzen, ob es sich um eine steigende oder eine fallende Flanke handelt. Bei Signalstörungen soll die Methode einen Wert > 1 zurückliefern. Bedenken Sie, dass zu kurze Flanken durch Signalrauschen bereits im Timer-Interrupt herausgefiltert werden. Für diesen Fall sind also nur zu lange Flanken relevant.
An dieser Stelle kann es hilfreich sein, sich zunächst zu skizzieren, welche Eingabekombinationen von langen und kurzen Flanken möglich sind und welchen Einfluss dies auf die Datenbits hat. Da es notwendig sein kann, dass im Signalverlauf Füllflanken eingefügt werden, muss die Funktion wissen, ob bei Ihrem vorhergehenden Aufruf schon eine Füllflanke gefunden wurde oder nicht. Beachten Sie dazu den Abschnitt über die

Manchester-Dekodierung in der Beschreibung des RFID-Treibers.
Sollte die Funktion `rfid_decodeManchester` einen Fehler im Signal finden (z. B. durch eine unzulässige Signalfolge) – also kein Datenbit und keine Füllflanke – muss der Tagpuffer zurückgesetzt werden.

Hinweis

Beachten Sie beim Einfügen der empfangenen Datenbits in die Variable `rfid_tagBuffer`, dass in C Literale bei der Anwendung von Bitoperationen implizit in eine 16 Bit Variable gecastet werden. Daher muss in diesen Fällen zunächst ein expliziter Typecast auf den gewünschten Typen (in diesem Fall `uint64_t`) durchgeführt werden.

Achtung

Die Verwendung des Debuggers kann die Aufzeichnung des Signals stören, wodurch die Zeitabstände der Signalflanken nicht immer korrekt sind. Daher raten raten wir von der Benutzung des Debuggers im Zusammenhang mit `rfid_decodeManchester` ab.

rfid_getData Implementieren Sie die Funktion `rfid_getData`. Diese Funktion überprüft ob die vorliegenden Paritäten korrekt sind. Sollte an dieser Stelle festgestellt werden, dass ein Fehler vorliegt, wird `0` zurückgegeben. Die jeweiligen Prüfungen können natürlich nur durchgeführt werden, wenn die dazu benötigte Anzahl an Zeichen schon empfangen wurde. Es ist zulässig, die Funktion `rfid_getData` erst dann aufzurufen, wenn 64 Bit empfangen wurden und ein Header an der richtigen Stelle erkannt wurde. Die Funktion `rfid_getData` gibt die empfangene ID zurück, falls alle Paritätsprüfungen ohne Fehler verlaufen sind, anderenfalls wird eine `0` zurückgegeben.

rfid_receive Implementieren Sie die Funktion `rfid_receive`. Ein möglicher Funktionsablauf ist durch den Pseudocode in Listing 6.2 gegeben. Es sind selbstverständlich noch andere Ansätze denkbar. Die zuvor implementierten Funktionen `rfid_decodeManchester` und `rfid_getData` werden von dieser Funktion aufgerufen. Denken Sie daran, in Abhängigkeit der Konstante `DISPLAY_RFID_TAG_AFTER_RECEIVE`, gegebenenfalls die ID auf dem Display auszugeben. Diese Funktion terminiert erst, wenn eine gültige ID gefunden wurde. Dies bedeutet, dass jeder Aufruf dieser Funktion eine gültige ID zurück liefert.

```
1 while (ID nicht ausgelesen oder ungültig) {
2   Warte bis neue Daten empfangen wurden (head vs. tail)
3   Nächste Flanke analysieren mit rfid_decodeManchester
```

```
4
5     if (Flanke entspricht einem Bit) {
6       Bit in Tag Puffer schreiben
7
8       Auf Header prüfen
9       if (Header okay) {
10        Signal dekodieren mit rfid_getData
11      }
12    }
13  }
14  ggf. ID auf LCD ausgeben
15  ID zurückgeben
```

Listing 6.2: Pseudocode für den Ablauf von `rfid_receive`.

Hinweise zur Implementierung von `rfid_receive`

- Geben Sie die Tag-ID ohne Header, Stoppbit oder Paritätsbits aus.
- Geben Sie die Tag-ID ziffernweise als Hexadezimalzahl aus.

6.4.3 Zugriffsrechte

Legen Sie einen Datentyp `UserType` an, welche die in Kapitel 6.3.4 genannten `OS_UT_*` Konstanten enthält.
Erstellen Sie ein Anwendungsprogramm, das anhand des zuletzt gelesenen Tags den aktuellen Benutzer identifiziert und entsprechende Zugriffsrechte setzt wie in Kapitel 6.3.4 beschrieben. Die Bedeutung der einzelnen Erlaubnisanfragen (`OS_PR_*`) können sie dem Kommentar zum entsprechenden `enum PermissionRequest` in der `os_user_privileges.h` entnehmen.
Legen Sie außerdem eine Reihe von Tags geeignet ab, sodass eine Zuordnung zu Benutzerklassen möglich ist. Hierzu gibt es diverse Möglichkeiten. Es müssen keine Benutzer zur Laufzeit hinzugefügt oder geändert werden können.

6.4.4 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_rfid`:
  - ☐ Typbenennung für `RFID_Tag`
  - ☐ Typbennenung für `ManchesterDecode`

- ☐ Vervollständigung mehrerer `defines` zur Ermittlung der Zeitkonstanten des RFID-Signals
- ☐ `ManchesterDecode rfid_decodeManchester(uint8_t level, uint16_t now_rec)`
- ☐ `uint64_t rfid_getData(void)`
- ☐ `uint64_t rfid_receive(void)`

☐ `os_user_privileges:`

- ☐ Anlegen der Datenstruktur `UserType`
- ☐ Anpassen von `AccessPermission os_askPermission(PermissionRequest pr, RequestArgument ra, RequestArgumentFlag raf, prog_char const** reason)`
- ☐ `void os_loginUser(UserType ut)`

☐ Anwendungsprogramm

- ☐ Erstellung eines Anwendungsprogrammes in `progs.c` zur Benutzersteuerung

6.5 Präsenzaufgaben

Für diesen Versuch erhalten Sie von den Betreuern einen Satz von Tags, sowie eine Zuordnung der auf diesen Tags gespeicherten IDs und der dazugehörigen Benutzerklassen.

6.5.1 Auslesen von Tags

Testen Sie die Funktionalität ihres RFID-Treibers, indem sie die IDs der Tags auf dem Display ausgeben lassen. Vergleichen Sie die ausgelesenen IDs mit den vorgegebenen IDs.

6.5.2 Zugriffsrechte

Legen Sie Benutzereinträge für die zur Verfügung gestellten Tags an und testen Sie die Benutzeridentifikation, indem Sie die Benutzerklasse der zuletzt angemeldeten Person auf dem Display ausgeben. Testen Sie außerdem die Zugriffskontrollmechanismen des Taskmanagers bei verschiedenen Benutzerklassen.

6.5.3 Weitere Benutzerklasse (optional)

Führen Sie eine weitere Benutzerklasse *Administrator* ein. Administratoren sollen tatsächlichen Vollzugriff auf den Taskmanager erhalten, ohne die Einschränkungen, die ein normaler Benutzer hat.

Achtung

Wenn ein Administrator den Taskmanager verlässt, darf er nicht automatisch abgemeldet werden, da ein neuer Login ggf. nicht möglich ist. Das Anmelden einer anderen Person soll einen Administrator ausloggen können.

6.6 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	frei
	B4	frei
	B5	frei
	B6	frei
	B7	frei
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	RFID-Platine (durchgeschliffen)
	D1	RFID-Platine (durchgeschliffen)
	D2	RFID-Platine (durchgeschliffen)
	D3	RFID-Platine (durchgeschliffen)
	D4	RFID-Platine (RDY/CLK)
	D5	RFID-Platine (MOD)
	D6	RFID-Platine (DEMOD_OUT)
	D7	RFID-Platine (SHD)

Pinbelegung für Versuch 6 (*RFID*).