UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TECHNISCHE INFORMATIK

Übungsblatt 5 (praktisch)

Space Invaders auf einem ATmega16

Vorlesung: Technische Grundlagen der Informatik 1, Sommersemester 2020

Dozent: Dr.-Ing. Kristian Ehlers

Wie auch die vorherigen Praktischen Übungen wird diese fünfte und letzte Praktische Übung in diesem Semester in Simulation durchgeführt. Um den Kontext zu waren, sind dennoch Bezüge zum Steckbrettsystem und dem realen Aufbau im Praktikumslabor in der Beschreibung enthalten. Sämtliche Peripherie wird im ITImega16-Simulator, wie bereits bekannt, bereitgestellt. Zudem dient der Simulator wieder als Testumgebung Ihres Codes. Laden Sie sich das Template aus dem Moodle herunter und ergänzen Sie es um Ihre Lösung.

Ziele und Umfeld des Versuchs

Eingebettete Systeme auf der Basis von Mikrocontrollern müssen in vielen Fällen in Echtzeit auf Ereignisse in ihrer Umgebung reagieren. Die Echtzeitforderung bedeutet, dass das System innerhalb eines definierten, kurzen Zeitraums auf das Eintreten eines Ereignisses reagieren muss. Es kann beispielsweise erforderlich sein, dass die Bewegung eines Roboterarms innerhalb von Bruchteilen einer Sekunde nach dem Auslösen einer Lichtschranke gestoppt wird. Um ein solches Verhalten zu gewährleisten, ist es häufig erforderlich, dass der Mikrocontroller bei Eintritt des Ereignisses (z. B. Lichtschranke unterbrochen) sofort die normale Programmausführung unterbricht und eine Befehlsfolge zur Behandlung des Ereignisses (z. B. Not-Stopp) abarbeitet.

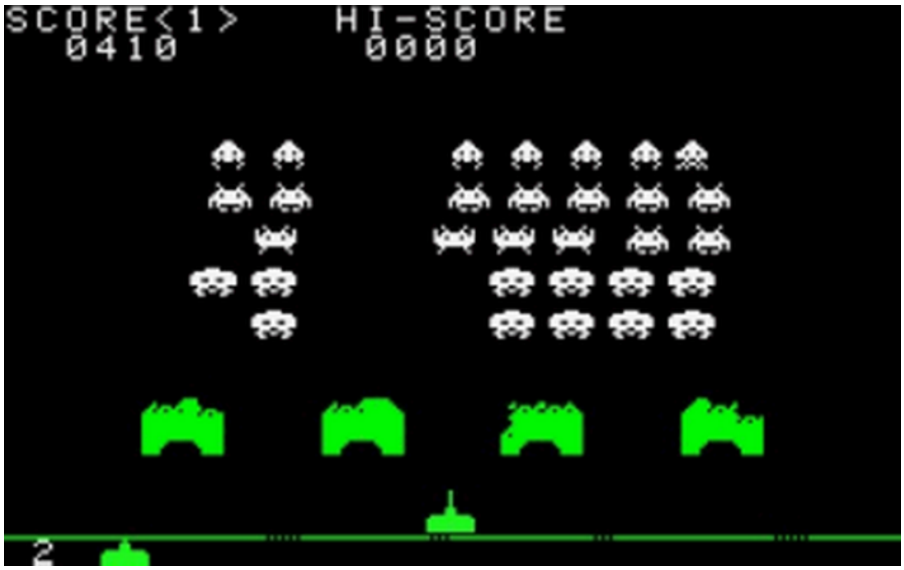


Abbildung 1: Space Invaders ist ein Shoot-Em-Up Spiel aus dem Jahr 1978, welches neben Pac-Man die frühe Entwicklung der Computerspiele geprägt hat. [http://de.wikipedia.org/wiki/Space_Invaders_(Computerspiel)]

Entsprechende Unterbrechungsmechanismen (Unterbrechung = engl. interrupt) stehen in Mikrocontrollern, wie dem im Praktikum verwendeten ATmega16, zur Verfügung. Neben externen Ereignissen, die dem Mikrocontroller über Spannungspegel an bestimmten Anschlüssen angezeigt werden, können Unterbrechungen auch durch Ereignisse innerhalb des Controllers ausgelöst

werden. So können etwa mit Hilfe der eingebauten Zeitgeber (engl. timer) in regelmäßigen Zeitabständen Unterbrechungen ausgelöst und Messwerte erfasst oder eine Uhrzeit weitergezählt werden.

Im Rahmen dieses Versuchs soll auf dem bekannten Steckbrettsystem ein kleines Spiel aufgebaut werden, welches dem Spiel Space Invaders aus dem Jahr 1978 ähnlich ist. Dabei wird ein über ein Potentiometer gesteuertes Raumschiff auf einem Zwei-Zeilen-Display hin und her bewegt und kann durch drücken eines Tasters zufällig auftauchende Alien-Schiffe von der Invasion abhalten.

In diesem Versuch werden alle grundlegenden Kenntnisse der ATmega Programmierung abgefragt. Die Positionierung des Raumschiffs wird durch eine Analog-Digital-Wandlung realisiert, das zeitliche Verhalten wird mit Hilfe von zwei Timern gesteuert und externe Interrupts werden bei Betätigung eines Tasters ausgeführt.

Einführung Timer

Timer/Counter0

Der ATmega16 besitzt die beiden 8-Bit-Zähler *Timer/Counter0* sowie *Timer/Counter2*. In diesem Versuch wird hiervon jedoch nur der *Timer/Counter0* verwendet. Er kann in einer Vielzahl von Betriebsarten verwendet werden, von denen die meisten nur in Zusammenhang mit der so genannten Pulsweitenmodulation (PWM) von Bedeutung sind. Vor dem Hintergrund dieses Versuchs sind nur die folgenden beiden Betriebsarten von Interesse:

(1) *Normal*:
Der Zähler zählt aufwärts bis 0xFF und springt im darauf folgenden Takt auf 0x00 über (Überlauf).

(2) *Clear Timer on Compare Match* (CTC):
Der Zähler zählt aufwärts bis einschließlich zu einem Vergleichswert und springt im darauf folgenden Takt auf 0x00 zurück.

Das Verhalten des *Timer/Counter0* wird durch den Inhalt verschiedener Register bestimmt. Die folgenden Abschnitte geben Aufschluss über die Funktion der einzelnen Bits in diesen Registern. Grau hinterlegte Felder sind in Zusammenhang mit diesem Versuch zu vernachlässigen.

Timer/Counter0 und Output Compare 0

Das Register Timer/Counter0 (`TCNT0`) speichert den aktuellen Zählerstand von Timer/Counter0 und kann sowohl ausgelesen als auch beschrieben werden. In dem Register Output Compare 0 (`OCR0`) kann ein Vergleichswert für den Zählerstand abgelegt werden. Abhängig von der Konfiguration des Timers, kann beim Erreichen eines Vergleichswerts ein Interrupt ausgelöst und eine Ausgabe über den Pin `OC0` erzeugt werden.

Timer/Counter0 Control Register

Der Zähler kann direkt mit dem Systemtakt getaktet werden oder alternativ, durch Verwendung des eingebauten Vorteilers (Prescalers), mit $clk_{I/O}/8$, $clk_{I/O}/64$, $clk_{I/O}/256$ oder $clk_{I/O}/1024$ des Systemtaktes. Darüber hinaus ist eine Taktung durch eine externe Quelle, d. h. durch ein von außen an den Mikrocontroller angelegtes Signal, möglich.
Zur Initialisierung des Timers werden die Register TCCR0 und TIMSK benötigt, welche im Folgenden nochmals illustriert werden:

Timer/Counter0 Control Register (TCCR0)

7	6	5	4	3	2	1	0
FOC0	**WGM00**	**COM01**	**COM00**	**WGM01**	**CS02**	**CS01**	**CS00**

COM01	COM00	Betriebsart von Pin OC0
0	0	OC0 getrennt (normale Betriebsart)
0	1	OC0 togglen bei Erreichen des Vergleichswerts
1	0	OC0 auf Low-Pegel bei Erreichen des Vergleichswerts
1	1	OC0 auf High-Pegel bei Erreichen des Vergleichswerts

WGM01	WGM00	Zählerbetriebsart
0	0	Normal: bis 0xFF zählen
1	0	CTC: Rücksetzen nach Erreichen des Vergleichswertes (OCR0)
andere Kombinationen		nicht relevante Einstellungen

CS02	CS01	CS00	Zählertakt
0	0	0	kein Zählertakt
0	0	1	$clk_{I/O}$ (= Systemtakt)
0	1	0	$clk_{I/O}/8$
0	1	1	$clk_{I/O}/64$
1	0	0	$clk_{I/O}/256$
1	0	1	$clk_{I/O}/1024$
1	1	0	ext. Pin T0 ↓
1	1	1	ext. Pin T0 ↑

Timer/Counter Interrupt Mask Register

Über das *Timer/Counter Interrupt Mask Register* (TIMSK) wird bestimmt, welche Ereignisse in den Timer/Countern zu einer Interruptanforderung führen.

Timer/Counter Interrupt Mask Register (TIMSK)

7	6	5	4	3	2	1	0
OCIE2	**TOIE2**	**TICIE1**	**OCIE1A**	**OCIE1B**	**TOIE1**	**OCIE0**	**TOIE0**

OCIE0	Interruptfreigabe für Timer/Counter0
0	kein Interrupt bei Erreichen des Vergleichswerts
1	Interrupt bei Erreichen des Vergleichswerts, wenn I-Flag gesetzt Adresse des Interrupt-Vektors: OC0addr

TOIE0	Interruptfreigabe für Timer/Counter0
0	kein Interrupt bei Überlauf
1	Interrupt bei Überlauf (0xFF → 0x00), wenn I-Flag gesetzt Adresse des Interrupt-Vektors: OVF0addr

Timer/Counter1

Neben den 8-Bit-Zählern *Timer/Counter0* und *Timer/Counter2* steht im ATmega16 mit dem *Timer/Counter1* ein 16-Bit-Zähler zur Verfügung. Er kann ebenso wie die 8-Bit-Zähler in einer Vielzahl von Betriebsarten verwendet werden. Ebenso wie beim Timer/Counter0 sind hier nur folgende beiden Betriebsarten von Interesse:

(1) *Normal*:
Der Zähler zählt aufwärts bis $FFFF und springt im darauf folgenden Takt auf $0000 über (Überlauf).

(2) *Clear Timer on Compare Match* (CTC):
Der Zähler zählt aufwärts bis einschließlich zu einem Vergleichswert und springt im darauf folgenden Takt auf $0000 zurück.

Das Verhalten des Timer/Counter1 wird ebenfalls durch den Inhalt verschiedener Register bestimmt. Die folgenden Abschnitte geben Aufschluss über die Funktion der einzelnen Bits in diesen Registern.

Timer/Counter1 und Output Compare 1 A/B (16-Bit-Register)

Das Register Timer/Counter1 (TCNT1) speichert den aktuellen Zählerstand von Timer/Counter1 und kann sowohl ausgelesen als auch beschrieben werden. In den Registern Output Compare 1 A und -1 B (OCR1A und OCR1B) können Vergleichswerte für den Zählerstand abgelegt werden. Abhängig von der Konfiguration des Timers kann beim Erreichen eines Vergleichswerts ein Interrupt ausgelöst, eine Ausgabe über die Pins OC1A bzw. OC1B erzeugt oder, nur im Falle von Vergleichswert A (OCR1A), der Zähler auf 0x0000 zurückgesetzt werden.
Da diese Register eine Breite von jeweils 16 Bit aufweisen, der ATmega16 jedoch lediglich über einen 8 Bit breiten Datenbus verfügt, müssen die Low- und High-Bytes der Register vom Assemblerprogramm aus einzeln angesprochen werden. Dabei ist eine Besonderheit zu beachten: Bei Schreibzugriffen muss unbedingt **zuerst das High-Byte** und **danach das Low-Byte** geschrieben werden, bei Lesezugriffen ist in umgekehrter Reihenfolge vorzugehen[1].

Timer/Counter1 (TCNT1) – 16-Bit-Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
High-Byte-Register **TCNT1H**								Low-Byte-Register **TCNT1L**							

Output Compare Register 1 A (OCR1A) – 16-Bit-Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
High-Byte-Register **OCR1AH**								Low-Byte-Register **OCR1AL**							

Output Compare Register 1 B (OCR1B) – 16-Bit-Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
High-Byte-Register **OCR1BH**								Low-Byte-Register **OCR1BL**							

[1] Zugriffe auf das High-Byte (z. B. TCNT1H) beziehen sich nicht direkt auf das jeweilige 16-Bit-Register (z. B. TCNT1), sondern auf ein 8 Bit breites „Schattenregister“. Der Inhalt dieses Schattenregisters wird nur beim Zugriff auf das Low-Byte (z. B. TCNT1L) mit den oberen acht Bits des 16-Bit-Registers synchronisiert. Auf diese Weise werden stets alle 16 Bits gleichzeitig ausgelesen oder beschrieben.

Timer/Counter1 Control Register A/B

Der Zähler kann direkt, wie auch schon beim Timer/Counter0, mit dem Systemtakt getaktet werden oder alternativ, durch Verwendung des eingebauten Vorteilers (Prescalers), mit $clk_{I/O}/8$, $clk_{I/O}/64$, $clk_{I/O}/256$ oder $clk_{I/O}/1024$ des Systemtaktes. Darüber hinaus ist eine Taktung durch eine externe Quelle, d. h. durch ein von außen an den Mikrocontroller angelegtes Signal, möglich.

Timer/Counter1 Control Register A (TCCR1A)

7	6	5	4	3	2	1	0
COM1A1	**COM1A0**	**COM1B1**	**COM1B0**	**FOC1A**	**FOC1B**	**WGM11**	**WGM10**

Timer/Counter1 Control Register B (TCCR1B)

7	6	5	4	3	2	1	0
ICNC1	**ICES1**	-	**WGM13**	**WGM12**	**CS12**	**CS11**	**CS10**

COM1*x*1	**COM1*x*0**	**Betriebsart von Pin OC1*x*** ($x \in \{A, B\}$)
0	0	OC1*x* getrennt (normale Betriebsart)
0	1	OC1*x* togglen bei Erreichen des Vergleichswerts *x*
1	0	OC1*x* auf Low-Pegel bei Erreichen des Vergleichswerts *x*
1	1	OC1*x* auf High-Pegel bei Erreichen des Vergleichswerts *x*

WGM13	**WGM12**	**WGM11**	**WGM10**	**Zählerbetriebsart**
0	0	0	0	Normal: bis $FFFF zählen
0	1	0	0	CTC: Rücksetzen des Zählers nach Erreichen des Vergleichswertes A (OCR1A)
andere Kombinationen				nicht relevante/unzulässige Einstellungen

CS12	**CS11**	**CS10**	**Zählertakt**
0	0	0	kein Zählertakt
0	0	1	$clk_{I/O}$ (= Systemtakt)
0	1	0	$clk_{I/O}/8$
0	1	1	$clk_{I/O}/64$
1	0	0	$clk_{I/O}/256$
1	0	1	$clk_{I/O}/1024$
1	1	0	ext. Pin T1 ↓
1	1	1	ext. Pin T1 ↑

Timer/Counter Interrupt Mask Register

Über das *Timer/Counter Interrupt Mask Register* (TIMSK) wird bestimmt, welche Ereignisse in den Timer/Countern zu einer Interruptanforderung führen.

Timer/Counter Interrupt Mask Register (TIMSK)

7	6	5	4	3	2	1	0
OCIE2	**TOIE2**	**TICIE1**	**OCIE1A**	**OCIE1B**	**TOIE1**	**OCIE0**	**TOIE0**

OCIE1*x*	**Interruptfreigabe** ($x \in \{A, B\}$) **für Timer/Counter1**
0	kein Interrupt bei Erreichen des Vergleichswerts
1	Interrupt bei Erreichen des Vergleichswerts *x*, wenn I-Flag gesetzt Adresse des Interrupt-Vektors: OC1*x*addr

TOIE1	Interruptfreigabe für Timer/Counter1
0	kein Interrupt bei Überlauf
1	Interrupt bei Überlauf (0xFFFF → 0x0000), wenn I-Flag gesetzt Adresse des Interrupt-Vektors: OVF1addr

Einführung externe Interrupts

In diesem Versuch beschäftigen Sie sich mit der Programmverzweigung als Reaktion auf einen externen Interrupt. Als Beispiel werden hier die beiden externen Interrupts `INT0` und `INT1` genauer beschrieben.

Allgemeine Funktionsweise

Der Interrupt `INT0` wird ausgelöst, wenn sich der an Pin PD2 anliegende Wert ändert, `INT1` reagiert auf Änderungen an Pin PD3.

Konfiguration

Wie alle speziellen Funktionen des ATmegas müssen auch die externen Interrupts initial konfiguriert werden, damit sie aktiviert und funktionsfähig sind. Die Funktionsweise wird im Register `MCUCR` des ATmegas eingestellt, beispielsweise ob die Interrupts bei einer steigenden Flanke (*low* nach *high*) oder bei einer fallenden Flanke (*high* nach *low*) ausgelöst werden. Dafür gibt es in diesem Register die Bits `ISC00`, `ISC01` (betreffen `INT0`) und `ISC10` und `ISC11` (betreffen `INT1`).

Hier eine Übersicht über die möglichen Einstellungen und was sie bewirken:

ISC11 oder ISC01	ISC10 oder ISC00	Beschreibung
0	0	Low-Level am Pin löst den Interrupt aus
0	1	Jede Änderung am Pin löst den Interrupt aus
1	0	Eine fallende Flanke löst den Interrupt aus
1	1	Eine steigende Flanke löst den Interrupt aus

Nach der Einstellung des Verhaltens müssen die Interrupts noch aktiviert werden. Dieses geschieht, in dem die Bits `INT0` bzw. `INT1` im Register `GICR` auf *high* gesetzt werden.

Wichtig:
Die Register `MCUCR` und `GICR` gehören zwar zu den IO-Registern, können aber nicht wie andere mit den Befehlen `cbi` und `sbi` verwendet werden. Diese Befehle wirken nur auf die IO-Register bis zur Adresse `0x1F`. Somit bleiben zum Zugriff auf diese Register nur die Befehle `in` und `out` übrig.

Schließlich muss man noch das Ausführen von Interrupts allgemein aktivieren, was man durch einfaches Aufrufen des Assemblerbefehls `sei` bewerkstelligt.

Beispiel:
In diesem Beispiel sehen sie die Konfiguration des `INT1` Interrupts als Reaktion auf eine steigende Flanke am Pin PD3 mit vorangestellter Interrupt-Vektor-Tabelle:

```
.cseg
.org   0x000
       rjmp   init
.org   INT1addr
       rjmp   INT1_ISR

init:
ldi    R16, ((1<<ISC11)|(1<<ISC10))
out    MCUCR, R16
ldi    R16, (1<<INT1)
out    GICR, R16
sei
```

Vorbereitungsaufgaben

Die folgenden Vorbereitungsaufgaben dienen dazu, die Aufgabe genau zu spezifizieren und Ihnen eine Schritt-für-Schritt Anleitung zur Lösung des Problems zu geben. Bearbeiten Sie die Aufgaben vollständig zu Hause, mit dem Ziel, beim Versuch in unserem Labor nur noch den lauffähigen Code eingeben zu müssen.

Aufgabenstellung – Spielprinzip

Bei dem Spiel Space Invaders geht es darum, die Invasion von Außerirdischen mithilfe eines Raumschiffes zurückzuschlagen. Abbildung 2 zeigt zwei Screenshots von der speziell für das 2-Zeilen Display konzipierten Version des Spiels. Es taucht für eine vorgegeben Zeit an einer zufälligen Position in der oberen Textzeile ein Außerirdischer auf. Die Aufgabe ist nun, mit dem Raumschiff den Außerirdischen zu vertreiben, indem es direkt unter den Außerirdischen bewegt und dieser dann mit dem Laserstrahl bestrahlt wird. Die Bewegung des Raumschiffs erfolgt mithilfe eines Potentiometers und das Abfeuern des Lasers wird durch das Betätigen eines Tasters ausgelöst. Für jeden vertriebenen Außerirdischen gibt es einen Punkt. Jedes Spiel dauert 60 Sekunden. Innerhalb dieser Zeit tauchen immer wieder neue Außerirdische auf und das Ziel ist es, möglichst viele Punkte zu erreichen. Natürlich hat man nicht unbegrenzt Energie für den Laser. Jeder Schuss kostet einen Energiepunkt. Für einen erfolgreich vertriebenen Außerirdischen erhält man zwei Energiepunkte zurück, jedoch ist die Anzahl an Energiepunkten auf 20 begrenzt. Das Spiel ist somit zu Ende, wenn die Zeit abgelaufen ist oder wenn die komplette Energie für den Laser verbraucht wurde. Auf der rechten Seite des Displays werden in der ersten Zeile die bereits verstrichene Zeit in Sekunden und in der zweiten Zeile die noch verbleibenden Energiepunkte angezeigt. Um das Spiel (neu) zu starten muss der Taster einmal betätigt werden.

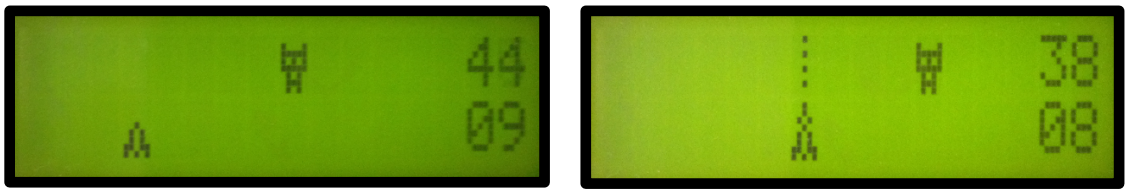


Abbildung 2: Screenshots des Spiels auf dem verwendeten 2-Zeilen Display.

Textmatrix-Display

Für die Ausgabe von alphanumerischem Text steht ein zweizeiliges Textmatrixdisplay (2x16 Zeichen) zur Verfügung.
In die Anzeigeeinheit integriert ist ein spezieller Controller vom Typ HD44780. Dieser Typ hat sich am Markt durchgesetzt und kann als Quasi-Standard für derartige Displays bezeichnet werden. Der Displaycontroller speichert die auszugebenden Zeichen und übernimmt die Ansteuerung der einzelnen Anzeigeelemente. Nach außen geführt sind ein 4 Bit breiter, bidirektionaler Datenbus (Anschlüsse D4 bis D7), die Kontrollsignale Enable (E1), Register Select (RS) und Read/Write (R/W) sowie getrennte Anschlüsse für die Energieversorgung von Anzeigeeinheit und Hintergrundbeleuchtung. Die Daten- und Kontrollsignale können direkt mit Ein-/Ausgabepins des ATmega16 verbunden werden. Die Energieversorgung der Anzeigeeinheit erfolgt unmittelbar über das Steckbrett, die Anschlüsse für die Beleuchtung sind hingegen von der Oberseite des Moduls zugänglich, durch ein LED-Symbol gekennzeichnet und mit A (Anode) und C (Kathode) beschriftet.

Ausgabe eines Zeichens auf dem Display

Um ein Symbol auf dem LC-Display auszugeben, sind zwei Informationen notwendig. Das Display muss wissen, an welcher Position, also in welcher Zeile [0-1] und welcher Spalte [0-15], das Symbol

dargestellt werden soll und natürlich, welches Symbol dargestellt werden soll. Deswegen erfolgt das Anzeigen in zwei Schritten.

1. Setzen des Cursors
   Um den Cursor an eine definierte Position des Displays setzen zu setzen, muss das Unterprogramm `lcd_setcursor` aufgerufen werden. Dieses erwartet in den unteren vier Bit des Registers R16 die Spalte, in die der Cursor bewegt werden soll [0-15]. Ist das Bit6 nicht gesetzt, so bewegt sich der Cursor innerhalb der oberen Zeile des Displays und ist Bit6 gesetzt, so springt der Cursor in die untere Zeile.
2. Darstellen des Symbols
   Um das Symbol darzustellen, muss das Unterprogramm `lcd_data` aufgerufen werden. Dieses erwartet im Register R16 den ASCII Code des darzustellenden Symbols. Um die selbst definierten Symbole anzuzeigen muss im Register R16 ein Wert zwischen null und sieben übergeben werden.
3. Das Anzeigen der neu gesetzten Informationen erfolgt über das Unterprogramm `lcd_redraw`

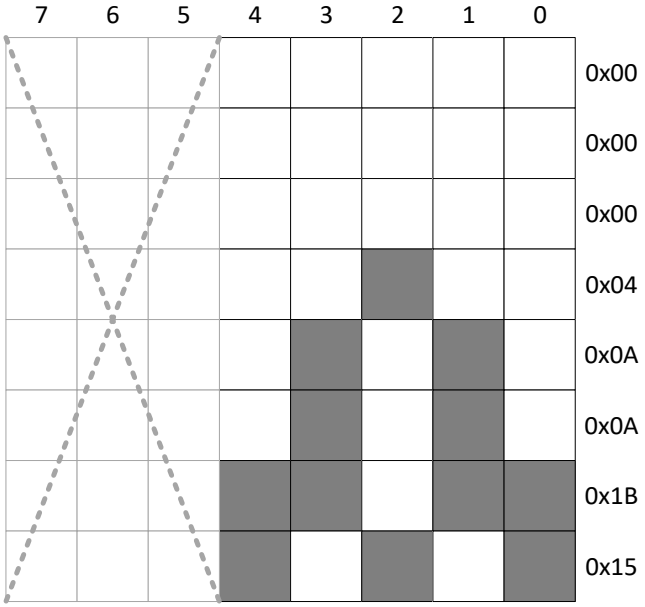


Abbildung 4: Design der Symbole

Design des Außerirdischen

Bei dem hier verwendeten 2-Zeilen Display besteht jede Zeile aus 16 Zeichen. Jedes Zeichen besteht aus 5x8 Pixeln, die entweder leuchten oder nicht. Es gibt die Möglichkeit, eigene Symbole zu erstellen, indem man für jede Zeile des Symbols den 8-Bit Zahlenwert angibt, der binär durch die gesetzten Pixel kodiert wird. In Abbildung 4 ist das Symbol des Raumschiffs mit den korrespondierenden Werten für jede Zeile dargestellt. Da jedes Symbol eine Breite von 5 Pixeln hat, werden die 3 hochwertigsten Bits nicht berücksichtig. Entwerfen Sie Ihr eigenes Symbol für einen Außerirdischen und bestimmen Sie die zu den 8 Zeilen korrespondierenden Zahlenwerte. Nutzen Sie dafür die Vorlage im Anhang der Versuchsbeschreibung.

Portbelegung

Vor der Erstellung des Programms sind die erforderlichen Schnittstellen zur Kommunikation mit der Prozessumwelt zu spezifizieren. Bei diesem Versuch sind folgende Ports des ATmega16 belegt:

ATmega I/O Port	I/O Richtung	Anschluss
PA0	Eingang	Potentiometer
PB2	Eingang	Taster
PD0	Ausgang	D4 (Display)
PD1	Ausgang	D5 (Display)
PD2	Ausgang	D6 (Display)
PD3	Ausgang	D7 (Display)
PD4	Ausgang	RS (Display)
PD5	Ausgang	E1 (Display)

Implementierung

Da die Implementierung des gesamten Spiels für einen Praktikumsversuch zu komplex ist, sind Ihnen im Template zu diesem Versuch einige Unterprogramme gegeben. Machen Sie sich mit dem gesamten Template vertraut. Beachten Sie, dass die gegebenen „Variablen“ im RAM abgelegt sind und diese von den Unterprogrammen genutzt werden. Um also eine korrekte Funktionalität gewährleisten zu können, müssen Sie diese „Variablen“ nutzen. Machen Sie sich weiterhin mit der grundlegenden Funktionsweise der einzelnen gegebenen Unterprogramme und mit den gegebenen Konstantendefinitionen vertraut. Es wird von Ihnen **nicht** verlangt, die **gegebenen** Unterprogramme in der SpaceInvaders.asm Datei bis ins kleinste Detail zu erklären. Sie sollten jedoch einen Überblick über die Funktionalität geben können.

Im Folgenden werden Ihnen Anleitungen und Hinweise zur Implementierung des Spiels gegeben. In Abbildung 5 ist eine Übersicht des Zusammenspiels der wichtigsten Unterprogramme, Interrupt Service Routinen und des Hauptprogramms dargestellt.

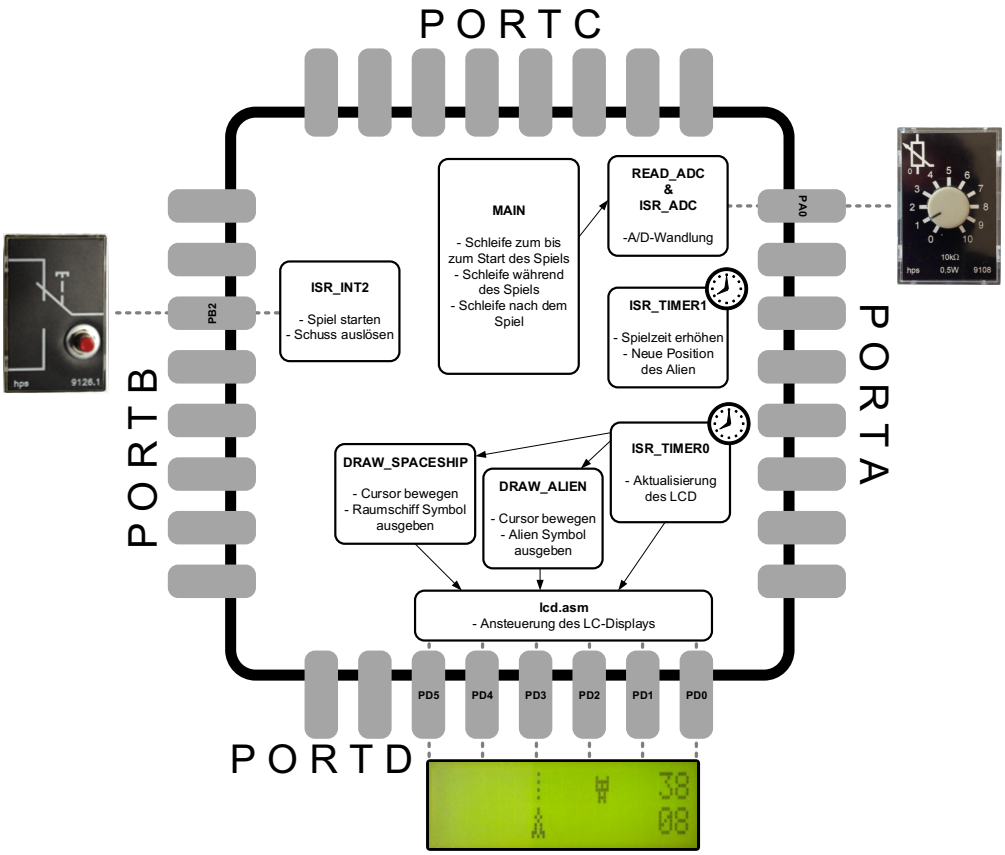


Abbildung 5: Übersicht des Zusammenspiels der wichtigsten Codeabschnitte, Unterprogramme und Interrupt Service Routinen. Die Spitze eines Pfeils weist jeweils auf das Unterprogramm oder den Quellcode der von einem anderen Teil des Programms genutzt bzw. aufgerufen wird.

SREG_GAME – Das Status-Register des Spiels

Das Spiel besteht aus mehreren Unterprogrammen. Um den Status des Spiels global zu speichern und im jedem Unterprogramm verfügbar zu machen, werden 8 Bit unter dem Label SREG_GAME im RAM abgelegt, von denen Bit0 bis Bit4 definierte Bedeutungen haben. Die restlichen Bits werden nicht genutzt. Reservieren Sie den benötigten Speicher. Der Aufbau des Statusregisters ist in Abbildung 6 dargestellt.

7	6	5	4	3	2	1	0
			F	H	A	T	S

Abbildung 6: SREG_GAME

Die einzelnen Bits haben folgende Bedeutung:

Start (S)	-	1 -> Spiel läuft
Timer (T)	-	1 -> Zeit ist abgelaufen
Ammo (A)	-	1 -> Munition ist leer
Hit (H)	-	1 -> der letzte Schuss war ein Treffer
Fail (F)	-	1 -> der letzte Schuss war kein Treffer

Eingabe der eigenen Alien Spezies

Im ROM an Adresse $30 sind unter dem Label `lcd_user_char` die acht benutzerdefinierten Symbole für das Raumschiff, die Explosion, den Schuss und die drei gegebenen Alien Spezies

abgelegt. Das letzte Symbol entspricht wieder dem Raumschiff. Da wir unser eigenes Raumschiff nicht verjagen wollen, ersetzen Sie bitte an der entsprechenden Stelle das Symbol um Ihre eigene Alien Spezies.

Allgemeine Initialisierungen

Nehmen Sie unter dem Label `init` bitte alle notwendigen Initialisierungen vor. Nutzen Sie dafür bitte die durch entsprechende Kommentare gekennzeichneten Zeilen im Template. Initialisieren Sie den Stackpointer, alle genutzten Ein- und Ausgabeports und die Anfangswerte der „Variablen". Für das Aktualisieren der Darstellungen auf dem Display wird alle rund 100 ms die Interrupt Service Routine `ISR_TIMER0` des Timer0 ausgeführt. Initialisieren Sie den Timer0, um die gewünschte Funktionalität zu erhalten. Es empfiehlt sich, den Timer im CTC Modus zu verwenden, einen zu erreichenden Zählerwert vorzugeben und den Timer mit einem Prescaler zu versehen. Aktivieren Sie den entsprechenden Interrupt des ATmega und nehmen Sie den Eintrag in der Interrupt Vektor Tabelle vor.

Das Raumschiff

Für das Ändern der Position des Raumschiffs wird ein am PINA0 angeschlossenes Potentiometer genutzt. Die anliegende Spannung muss mithilfe des AD-Wandlers digitalisiert und später in die Position umgerechnet werden.

Das Ihnen gegebene Unterprogramm `read_adc` realisiert die A/D-Wandlung und liefert im Register R16 die oberen acht Bit des aktuell ermittelten A/D Wertes zurück.

Realisieren Sie bitte im Unterprogramm `convert_value_to_pos` die Berechnung der aktuellen Position des Raumschiffs anhand des im Register R16 an das Unterprogramm übergebenen AD-Wertes. Das Raumschiff befindet sich stets in der zweiten Zeile des Displays an einer Position zwischen 0 und 13. Die ermittelte Position soll im Register R16 zurückgegeben werden.

Das Unterprogramm `draw_spaceship` ist für das Anzeigen des Raumschiffs und des eventuell getätigten Fehlschusses auf dem Display verantwortlich. Zu diesem Zweck setzt das Unterprogramm den Cursor an die Position des Raumschiffs (`SPACESHIP_POSITION`) und gibt anschließend das in der „Variablen" `SPACESHIP` gespeicherte Symbol des Raumschiffs aus. Um kleinere Animationen, wie den Schuss des Raumschiffs zu ermöglichen, gibt es für das Raumschiff zwei Symbole; einmal mit und einmal ohne angedeuteten Schuss. Aus diesem Grund muss geprüft werden, ob das ausgegebene Symbol dem des Raumschiffs mit Schuss (`SYMBOL_SPACESHIP_FIRE`) entspricht. Ist das der Fall, so muss das Symbol auf das Standard Symbol des Raumschiffs (`SYMBOL_SPACESHIP`) zurückgesetzt werden. Weiterhin muss in diesem Unterprogramm geprüft werden, ob der letzte Schuss daneben ging (F Flag im `SREG_GAME` ist gesetzt). Ist dies der Fall, so müssen in der oberen Zeile an der Position des Raumschiffs des Symbol `SYMBOL_FIRE` ausgegeben und das F Flag zurückgesetzt werden.

Das Alien

Das Unterprogramm `draw_alien` setzt den Cursor an die Position des Aliens (`ALIEN_POSITION`) in der oberen Zeile des Displays und stellt das Symbol des Aliens (`ALIEN_SPECIES`) dar.

Die Zeit

Eine Runde des Spiels läuft maximal 60 Sekunden. Für die Zeitmessung wird der Timer1 genutzt. Nehmen Sie wie auch schon beim Timer0 alle notwendigen Initialisierungen vor, um die Interrupt Service Routine des Timer1 `ISR_TIMER1` jede Sekunde aufzurufen.

Implementieren Sie die Funktionalität der Interrupt Service Routine `ISR_TIMER1`. Diese soll nur komplett ausgeführt werden, wenn das Spiel läuft (S Flag des `SREG_GAME` ist gesetzt) und in diesem

Fall die in der „Variablen“ TIME gespeicherte Zeit inkrementieren. Sind die 60 Sekunden abgelaufen, so muss komplette SREG_GAME gelöscht und nur das T Flag gesetzt werden.

Da sich das Alien jede Sekunde an eine zufällige Position begeben soll, muss die ISR die Position des Aliens ändern. Mit dem Unterprogramm rand kann eine neue Position ermittelt und die „Variable“ ALIEN_POSITION aktualisiert werden.

Sollte der letzte Schuss ein Treffer gewesen sein (H Flag im SREG_GAME ist gesetzt), so soll zudem eine andere Spezies auftauchen. Hierfür muss lediglich das Unterprogramm new_alien_species aufgerufen und im Anschluss das H Flag zurückgesetzt werden.

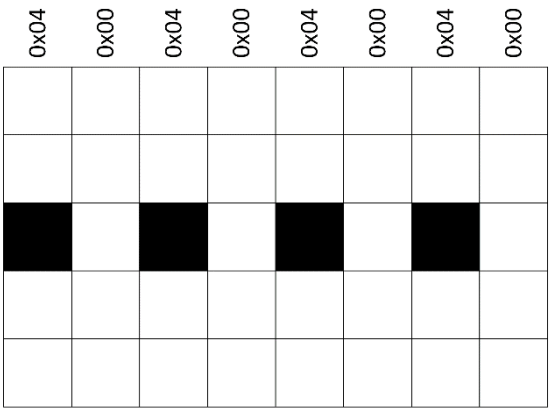
Der Taster

Mit dem Taster kann das Spiel gestartet und während des Spiels ein Schuss ausgelöst werden. Bei jedem Betätigen des Tasters soll auf der fallenden Flanke der Interrupt2 ausgelöst und die Interrupt Service Routine ISR_INT2 aufgerufen werden. Nehmen Sie alle notwendigen Initialisierungen vor und Implementieren Sie die Interrupt Service Routine ISR_INT2.

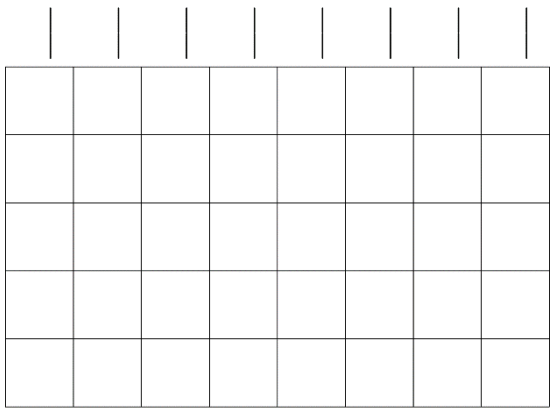
Sollte das Spiel nicht laufen (S Flag des SREG_GAME ist nicht gesetzt), so wird das S Flag gesetzt und die ISR verlassen. Läuft hingegen das Spiel, sind einige Dinge abzuarbeiten. Die Betätigung des Taster löst in diesen Fällen eine Schuss aus. Das Symbol des Raumschiffs in der Variablen SPACESHIP muss auf SYMBOL_SPACESHIP_FIRE geändert werden. Im Anschluss muss geprüft werden, ob der Schuss ein Treffer war oder nicht. Sollte getroffen worden sein, so muss das entsprechende Bit im SREG_GAME gesetzt werden. Weiterhin werden sowohl die Punktzahl als auch die Munition inkrementiert. Beachten Sie dabei, dass der Munitionsvorrat den Wert 20 nicht überschreiten darf. Ein getroffenes Alien erkennt man an einem anderen Symbol. Bitte setzen Sie im Falle eines Treffers das Symbol des Aliens in der Variablen ALIEN_SPECIES auf EXPLOSION. Sollte der aktuelle Schuss kein Treffer gewesen sein, so muss lediglich das F Flag des SREG_GAME gesetzt und die Munition um eins verringert werden. Sollte die Munition ausgegangen sein, so soll das gesamte SREG_GAME gelöscht und nur das A Flag gesetzt werden.

Anhang

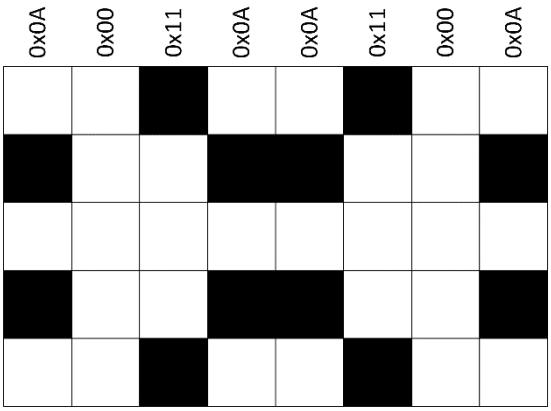
Schuss



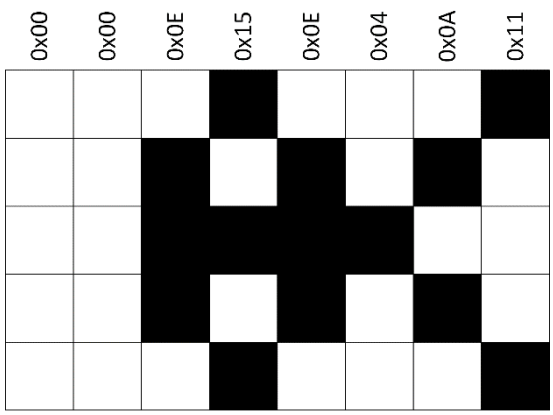
Eigene Spezies



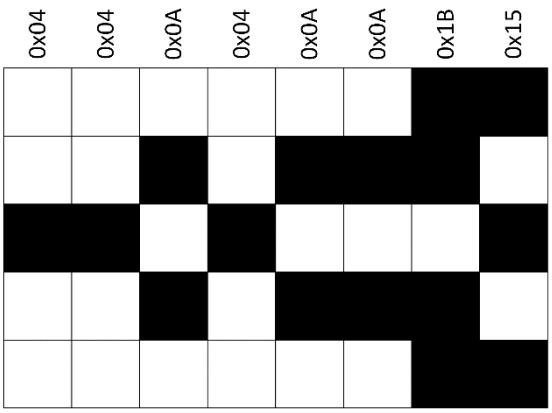
Explosion



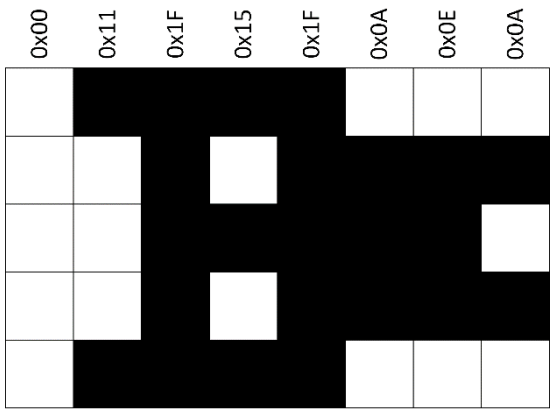
Alien Spezies 3



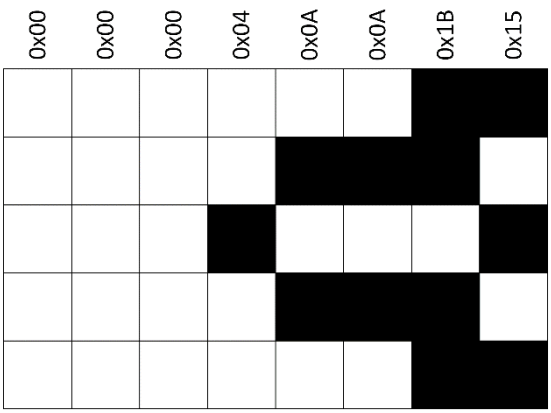
Raumschiff schießt



Alien Spezies 2



Raumschiff



Alien Spezies 1