U2 2. Übung

- SPiC-Boards
- Nachtrag zur Vorlesung
- Aufgabe 2

U2-1 SPiC-Boards

- Es wird wieder möglich sein SPiC-Boards selber zu bauen
  - Drei Lötabende in den Wochen nach dem Berg (Anfang Juni)
  - Die Boards können auch einzeln gekauft werden (zum selber Löten)
  - Kosten: 5 bis 10 Eur
  - Version 2.1: Kleine Verbesserungen
- Neu: Programmierer für den Parallelport
- Stimmungsbild:
  - SPiC-Board zum selber bauen
  - SPiC-Board am Lötabend bauen
  - Sollen USB->Parallel Umsetzer zentral besorgt werden (~5 Eur)?

U2-2 Nachtrag zur Vorlesung

1 Variablentypen

- Standardtypen sind in C nicht genau definiert:
  - `char` ist immer 8 Bit (bis auf einige Ausnahmen)
  - `short int` ≤ 16 Bit ≤ `int` ≤ `long int` ≤ `long long int`
  - Beispiel für drei Architekturen (Angaben in Bits)

	8-bit AVR	Intel x86-32	Intel x86-64
`char`	8	8	8
`short`	16	16	16
`int`	16	32	32
`long`	32	32	64

- Der genaue Wertebereich steht in der `limits.h`
  - z.B. `INT_MIN`, `INT_MAX` oder `UINT_MIN`, `UINT_MAX`

1 Variablentypen (2)

- Besser: Verwendung der `stdint.h`
  - `int16_t` 16 Bit mit Vorzeichen:
    `INT16_MIN` (–32,768) bis `INT16_MAX` (32,767)
  - `uint16_t` 16 Bit ohne Vorzeichen
    `UINT16_MIN` (0) bis `UINT16_MAX` (65,535)
  - Zeiger sind immer vom Adressbus der Architektur abhängig
    - AVR: 16 Bit
    - Aber: 8-Bitter -> Es kann immer nur ein Byte aus dem Speicher gelesen, bzw. geschreiben werden
- Vorteile:
  - Wertebereich ist bekannt
  - Insbesondere in der hardwarenahen Programmierung braucht man oft Datentypen einer bekannten, festen Größe (I/O-Register)
  - Quellcode kann leichter auf andere uC portiert werden

2 Lebensdauer von Variablen

- Die Lebensdauer einer Variablen bestimmt, wie lange der Speicherplatz für die Variable aufgehoben wird
- Zwei Arten
  - Speicherplatz bleibt für die gesamte Programmausführungszeit reserviert
    - ➤ statische (`static`) Variablen
  - Speicherplatz wird bei Betreten eines Blocks reserviert und danach wieder freigegeben
    - ➤ dynamische (`auto`) Variablen

2 Lebensdauer von Variablen (2)

auto-Variablen

- Alle lokalen Variablen sind automatic-Variablen
  - ➤ der Speicher wird bei Betreten des Blocks / der Funktion reserviert und bei Verlassen wieder freigegeben
    - ➥ der Wert einer lokalen Variablen ist beim nächsten Betreten des Blocks nicht mehr sicher verfügbar!
- Lokale auto-Variablen können durch beliebige Ausdrücke initialisiert werden
  - ➤ die Initialisierung wird bei jedem Eintritt in den Block wiederholt
  - !!! **wird eine auto-Variable nicht initialisiert, ist ihr Wert vor der ersten Zuweisung undefiniert** (= irgendwas)

2 Lebensdauer von Variablen (3)

static-Variablen

- Der Speicher für alle globalen Variablen ist generell von Programmstart bis Programmende reserviert
- Lokale Variablen erhalten bei Definition mit dem Schlüsselwort `static` eine **Lebensdauer über die gesamte Programmausführung** hinweg
  - ➥ der Inhalt bleibt bei Verlassen des Blocks erhalten und ist bei einem erneuten Eintreten in den Block noch verfügbar
  - !!! Das Schlüsselwort `static` hat bei globalen Variablen eine völlig andere Bedeutung (Einschränkung des Zugriffs auf das Modul)
- Static-Variablen können durch beliebige konstante Ausdrücke initialisiert werden
  - ➤ die Initialisierung wird nur einmal beim Programmstart vorgenommen (auch bei lokalen Variablen!)
  - • erfolgt keine explizite Initialisierung, wird automatisch mit 0 vorbelegt

U2-3 Aufgabe 2: snake

- Schlange bestehend aus benachbarten LEDs
- Länge 1 bis 5 LEDs, regelbar mit Potentiometer (POTI)
- Geschwindigkeit abhängig von Umgebungshelligkeit
- Bewegungsrichtung umschaltbar mit Taster

U2-4 Parameter der Schlange

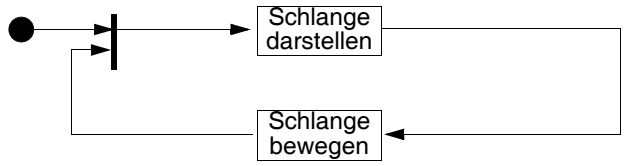
- Position des Kopfes
  - Nummer einer LED
  - Wertebereich [0; 7]
- Länge der Schlange
  - Ganzzahl im Bereich [1;5]
- Richtung der Schlange
  - aufwärts oder abwärts
  - z.B. 0 oder 1
- Geschwindigkeit der Schlange
  - hier: Durchlaufzahl der Warteschleife

U2-5 Zerlegung in Teilprobleme

- Basisablauf: Welche Schritte wiederholen sich immer wieder?
- Teilprobleme *können* in eigene Funktionen ausgelagert werden
- Wiederkehrende Teilprobleme *sollten* in Funktionen ausgelagert werden
- Welcher Zustand muss über Basisabläufe hinweg erhalten bleiben?
  - ist Zustand ggf. nur für ein Teilproblem relevant?
  - Sichtbarkeit dann auf das Teilproblem einschränken (Kapselung)

U2-5 Basisablauf

- Darstellung der Schlange
- Bewegung der Schlange



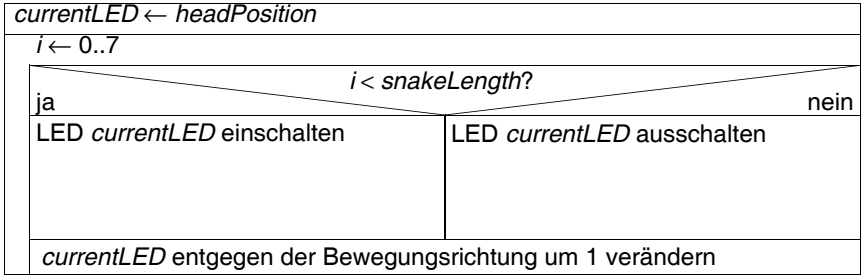
1 Darstellung der Schlange

- Bestimmung der Darstellungsparameter
  - Kopfposition
  - Länge
  - Richtung
- Anzeige der Schlange abhängig von den Parametern
  - Aktivieren der zur Schlange gehörenden LEDs
  - Deaktivieren der restlichen LEDs

2 Bewegung der Schlange

- Bestimmung der Bewegungsparameter
  - Geschwindigkeit
  - Richtung
- Bewegen der Schlange
  - Anpassen der Kopfposition abhängig von der Richtung
- Wartepause abhängig von der Geschwindigkeit
- ggf. Richtungsänderung
  - bisheriger Schlangenschwanz wird zum Schlangenkopf

U2-6 Ablaufplan für Teilprobleme Schlangenanzeige



U2-7 Bestimmung der Richtungsänderung

- Der Taster muss periodisch abgefragt werden
  - man bekommt nur eine Momentaufnahme des Tasters
- evtl. wird ein Tastendruck verpasst
  - Hochfrequente Abfrage zur Minimierung des Risikos
  - Wo im Programm ist dies sinnvoll möglich?
    - in der Warteschleife (jedem Durchlauf)
    - hier wird die meiste Zeit verbracht
- mehrfache Abfrage des Tasterzustands während eines Tasterdrucks
  - Mehrfachinterpretation eines Tasterdrucks vermeiden
  - Loslassen des Tasters muss explizit registriert werden
    - Historie über den vorigen Tasterzustand mitführen (Flankenerkennung)