Aufgabenblatt 7 Höhere Programmiersprachen

1 Bitmanipulation in C

1.1 Datentypen

Die Standarddatentypen in C/C++ wie ***unsigned int***, ***long int*** usw. haben je nachdem für welches Architekturmodel sie kompiliert worden sind unterschiedliche Bitbreiten. Aus diesem Grund wurden in C99 neue Datentypen definiert, deren Bitbreite von der verwendeten Architektur unabhängig ist. Diese Datentypen sind in der <stdint> Headerdatei definiert. Der Name der Datentypen besteht aus der Abkürzung ***int*** (für vorzeichenbehaftete), oder ***uint*** (für vorzeichenlose) gefolgt von der gewünschten Bitbreite, sowie dem suffix _t (für „Type“). Somit ist z.B. der ***uint8_t*** Datentyp immer ein vorzeichenloser 8 Bit Datentyp. Bei ***int16_t*** handelt es sich dann analog dazu um einen vorzeichenbehafteten 16 Bit Datentyp. Diese Datentypen erlauben eine bessere Portierung der Software zwischen Architekturen mit unterschiedlichen Bitbreiten, sowie bessere Speicherauslastung und sind in der Mikrocontrollerwelt zu bevorzugen.

1.2 Masken

Masken in C/C++ , soweit diese konstant sind, werden als Präprozessor Konstanten definiert

#define MY_VALUE 8

#define MY_OTHER_VALUE 9

Sie können wie gewohnt miteinander verkettet und/oder kombiniert werden:

something = MY_VALUE | MY_OTHER_VALUE;

something_else = MY_VALUE * MY_OTHER_VALUE;

something_shift_seven = MY_VALUE « 7

1.3 Aufgabenstellung

Gegeben sind folgende Konstanten (und zwar dort, wo die Konstanten in der Sprache c hingehören):
CONSTANT_A
CONSTANT_LOW
CONSTANT_HIGH

Ausgehend von der Konstanten A laden Sie folgende Werte (alle Indexe in Informatikerzählweise), falls nicht anders vermerkt, in 32 Bit breite ***uint*** Variablen:

- value_a – soll nur eine 8 Bit Variable sein. Sie enthält den Wert des dritten Nibbels.
- value_b – enthält den inversen Binären Wert (kein exklusives oder!)
- value_c – der höchste Nibble ist invertiert, ansonsten sind alle anderen Nibbels unverändert.
- value_d – alle **Bits**, bis auf die Nibbels zwei und drei sollten auf 1 gesetzt werden, die Nibbels zwei und drei sollten unverändert bleiben.

Ausgehend von den Konstanten Low und High:

- Packen Sie die beiden Konstanten als zwei 8 Bit Werte in eine 16 Bit uint ***Big Indian Variable***.
- Packen Sie die beiden Konstanten als zwei 16 Bit Werte in eine 32 Bit uint ***Big Indian Variable***.
- Packen Sie die beiden Konstanten als zwei 8 Bit Werte in eine 16 Bit uint ***Little Indian Variable***.

Diese Aufgabe kann auch in Online GDB gelöst und präsentiert werden.

Skript:
4.7 Zahlen Schieben
4.9 Binäre Operationen
4.10 Bitmasken definieren

2 Aufgabe 2 : GPIO in C

2.1 Hardware Register

Die Hardwareregister des Mikrocontrollers sind in der „LPC21XX.H“ Datei definiert. Sie können diese direkt wie Variable verwenden und zwar sowohl beim Schreiben wie auch beim Lesen von Werten z.B:

- ***IODIR0 = OUTPUT_MASKE*** – setze alle Bits, die in der der ***OUTPUT_MASKE*** „High“ sind, als Ausgänge.
- ***input = IOPIN0*** – kopiere den Zustand aller Eingangspins des Ports 0 in die ***input*** Variable

2.2 Logische Aufwertung bei Kontrollstrukturen in C

Bevor der Parameter einer ***if, while*** oder ***do...while*** Anweisung ausgewertet wird, wird dieser in einem Boolean Wert umgewandelt. Ist das Ergebnis des Parameters 0 so wird dieses als ***unwahr (false)*** gewertet, ist es wiederum ungleich 0 so wird dieser Parameter als ***wahr (true)*** gewertet. Schauen wir uns das am Beispiel einiger Anweisungen an:

- ***if (IOPIN & BIT_MASKE)*** – sobald einer der Bits in der Maske mit dem des Register übereinstimmt, wird die ***if*** Anweisung als wahr interpretiert.
- ***if ((IOPIN & BIT_MASKE) == BIT_MASKE)*** – in diesem Fall erfolgt die boolesche Evaluation erst nach dem Vergleich, so dass hier alle Bits aus der BIT_MASKE im Register gesetzt sein müssen, damit die ***if*** Anweisung als wahr ausgewertet wird.
- ***if (5)*** – immer wahr
- ***if (0)*** – immer unwahr
- ***while(1)*** – Workaholic
- ***while(0)*** – Arbeitsverweigerer
- ***while (counter)*** – laufe bis Zähler == 0 (counter = 5 → {5,4,3,2,1})
- ***while (counter+1)*** – laufe bis Zähler == -1 (counter = 5 → {5,4,3,2,1,0}), funktioniert nur mit ***int*** Variablen, bei ***uint*** Variablen ergibt das eine Endlosschleife !

2.3 Aufgabenstellung

Portieren Sie Ihr Programm aus der Aufgabe 5.2 in die Sprache C wobei sie jetzt 100% der Phase verwenden sollten. Versuchen Sie Ihr Programm, durch die geeignete Dimensionierung der Variablen auf Speicherverbrauch zu optimieren. Damit ist explizit ***nicht*** die Wiederverwendung von im Moment nicht benötigten Variablen gemeint, das ist die Aufgabe des Compilers/Optimierers. Beachten Sie bitte, dass Sie jetzt ***sinnvolle*** Variablennamen verwenden können (lese – müssen) also z.B. ***counter*** statt ***r0*** etc. Implementieren Sie dabei Ihre Assembler ***delay()*** Funktion als eine eigene Funktion in C.

Skript:
3.8 Hardware Register in C verwenden.

3 Umschaltbares Lauflicht

Erweitern Sie ihr Lauflicht um ein zweites alternatives Lauflicht. Jedes dieser Lauflichter ist in einer eigenen Funktion zu implementieren.

- Die Umschaltung der Lauflichter erfolgt über Tasten (Tastenbelegung siehe unten)
- Für die Abfrage der Tasten realisieren Sie bitte eine eigene Funktion. Alternativ können Sie für jede Umschaltfunktion eine eigene (***inline***) Funktion erstellen (z.b. ***_isStart()***).
- Falls sie für die Abfrage aller Tasten nur eine Funktion erstellen, so kann dieser auch nur einem numerischen Wert zurückgeben z.B:
  0 – ACTION_IDLE (Keine Taste gedrückt)
  1 – ACTION_START (Taste 0 gedrückt)
  …
  x – ACTTION_SWITCH (Tasten 0 & 1 sind gedrückt)

- Beachten Sie, das durch die Verwendung von zwei Tasten gleichzeitig eine bestimmte Abfragepriorität notwendig ist
- Die Tasten müssen (sollen) nicht sofort, sondern erst bei Schrittwechsel reagieren.
- In Pause- oder Stopp-Modus kann das ***delay()*** wie gewöhnlich periodisch aufgerufen werden.
- Es reicht, bzw. es ist die bessere, bevorzugte Version, wenn das ***delay()*** an nur einer Stelle im Programmcode ausgeführt wird.

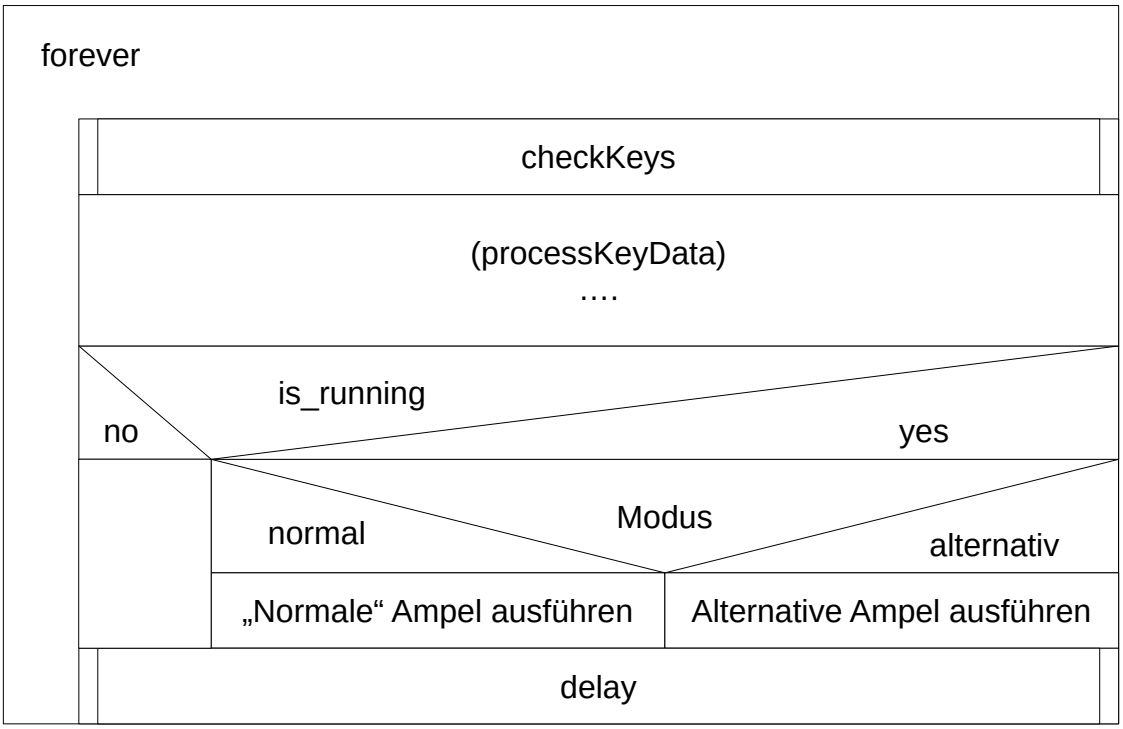

Abbildung 1: Möglicher Lösungsweg

Funktion der Tasten
Button 0 – Start
Button 1 – Pause / Continue
Button 3 – Reset
Button 0 & Button 1 – Lauflicht umschalten ohne Reset

Skript:
4.7 Zahlen Schieben
4.9 Binäre Operationen
4.10 Bitmasken definieren