Anwenderdokumentation
intstr V0.0.0

Integer to String

Wandelt Integer oder Integer mit Festkommaarithmetik in Strings um. Dabei wird ausschliesslich eine dezimale Darstellung verwendet.

Autor: bestucki
Datum: 07.04.2013
Version: V00

Versionsgeschichte

Version	Änderung	Name	Datum
V00	Erste Fassung	bestucki	07.04.2013

Inhaltsverzeichnis

1 Beschreibung der Bibliothek

Diese Bibliothek ist dazu da, um Integer oder Integer mit Festkommaarithmetik in Strings umzuwandeln. Dabei kann, anders als bei den allgemein geläufigen Funktionen itoa und utoa, die zu verwendende Basis nicht ausgewählt werden. Es wird ausschliesslich eine dezimale Darstellung verwendet. In vielen Fällen wird auch nicht mehr als das benötigt, daher wurde diese Zusatzfunktion zu Gunsten der Geschwindigkeit weggelassen.
Die in dieser Bibliothek enthaltenen Funktionen können auf Mikrocontrollern schneller als itoa und utoa sein (getestet mit dem PIC18F452-I/P und dem XC8-Compiler, beide von Microchip), vor allem, wenn 8 Bit Variablen umgewandelt werden müssen. Vergleiche zum Programmspeicherbedarf wurden keine durchgeführt.

Die Bibliothek wurde für Mikrocontroller entwickelt, kann natürlich auch auf einem PC oder einem anderen System verwendet werden.

1.1 Öffentliche Funktionen

In diesem Kapitel werden alle Funktionen beschrieben, die von einem Anwender/Programmierer direkt aufgerufen werden können.

1.1.1 Allgemeine Bemerkungen

Falls ein Datentyp, der in der Bibliothek verwendet wird (z.B. int24_t oder int64_t), vom verwendeten Compiler nicht unterstützt wird, so werden diese Funktionen nicht kompiliert und können somit auch nicht verwendet werden. Die dafür nötigen Informationen werden aus stdint.h entnommen.

Bei einer Verwendung auf einem Mikrocontroller, sollte immer der kleinstmögliche Datentyp verwendet werden, da z.B. eine Umwandlung eines int8_t mit int32str erheblich länger dauert als mit int8str.

Wenn der verwendete Compiler unbenutzten Code nicht automatisch entfernt, sollten nicht benutzte Funktionen manuell auskommentiert werden. So kann vor allem bei Mikrocontrollern wertvoller Speicherplatz gespart werden.

1.1.2 intXXstr

Prototyp:	`char *int8str(char *target, int8_t source);`
	`char *int16str(char *target, int16_t source);`
	`char *int24str(char *target, int24_t source);`
	`char *int32str(char *target, int32_t source);`
	`char *int64str(char *target, int64_t source);`	

Beschreibung:	Wandelt einen vorzeichenbehafteten Integer in einen String um.	
Übergabewerte:	`char *target`	Zeiger auf den String, in dem der umgewandelte Integer gespeichert werden soll.

	`int8_t source`
	`int16_t source`
	`int24_t source`
	`int32_t source`
	`int64_t source`
		Integer, der in einen String umgewandelt werden soll.

Rückgabewert:	`char *`	Zeiger auf den String target (vgl. Übergabewerte)
Empfohlene Verwendung:	`int16str(target,source);` `strRaglin(int16str(str,source),minlenght);`	
Bemerkungen:	keine	

1.1.3 uintXXstr

Prototyp:	`char *uint8str(char *target, uint8_t source);`
	`char *uint16str(char *target, uint16_t source);`
	`char *uint24str(char *target, uint24_t source);`
	`char *uint32str(char *target, uint32_t source);`
	`char *uint64str(char *target, uint64_t source);`	

Beschreibung:	Wandelt einen nicht vorzeichenbehafteten Integer in einen String um.	
Übergabewerte:	`char *target`	Zeiger auf den String, in dem der umgewandelte Integer gespeichert werden soll.

	`uint8_t source`
	`uint16_t source`
	`uint24_t source`
	`uint32_t source`
	`uint64_t source`
		Integer, der in einen String umgewandelt werden soll.

Rückgabewert:	`char *`	Zeiger auf den String target (vgl. Übergabewerte)
Empfohlene Verwendung:	`uint16str(target,source);` `strRaglin(uint16str(str,source),minlenght);`	
Bemerkungen:	keine	

1.1.4 fixintXXstr

Prototyp:	`char *fixint8str(char *target, int8_t source, int_fast8_t cp);`
	`char *fixint16str(char *target, int16_t source, int_fast8_t cp);`
	`char *fixint24str(char *target, int24_t source, int_fast8_t cp);`
	`char *fixint32str(char *target, int32_t source, int_fast8_t cp);`
	`char *fixint64str(char *target, int64_t source, int_fast8_t cp);`	

Beschreibung:	Wandelt einen vorzeichenbehafteten Integer in einen String um. Dabei wird der umzuwandelnde Integer mit 10^{-cp} multipliziert. Somit ist es möglich, Zahlen mit Festkommaarithmetik darzustellen,	
Übergabewerte:	`char *target`	Zeiger auf den String, in dem der umgewandelte Integer gespeichert werden soll.

	`int8_t source`
	`int16_t source`
	`int24_t source`
	`int32_t source`
	`int64_t source`
		Integer mit Festkommaarithmetik, der in einen String umgewandelt werden soll.

	`int_fast8_t cp`	Source wird mit 10^{-cp} multipliziert.
Rückgabewert:	`char *`	Zeiger auf den String target (vgl. Übergabewerte)
Empfohlene Verwendung:	`fixint16str(target,source,cp);` `strRaglin(fixint16str(str,source,cp),minlenght);`	
Bemerkungen:	keine	

1.1.5 fixuintXXstr

Prototyp:	`char *fixuint8str(char *target, uint8_t source, int_fast8_t cp);`
	`char *fixuint16str(char *target, uint16_t source,`
	`int_fast8_t cp);`
	`char *fixuint24str(char *target, uint24_t source,`
	`int_fast8_t cp);`
	`char *fixuint32str(char *target, uint32_t source,`
	`int_fast8_t cp);`
	`char *fixuint64str(char *target, uint64_t source,`
	`int_fast8_t cp);`	

Beschreibung:	Wandelt einen nicht vorzeichenbehafteten Integer in einen String um. Dabei wird der umzuwandelnde Integer mit 10^{-cp} multipliziert. Somit ist es möglich, Zahlen mit Festkommaarithmetik darzustellen,	
Übergabewerte:	`char *target`	Zeiger auf den String, in dem der umgewandelte Integer gespeichert werden soll.

	`uint8_t source`
	`uint16_t source`
	`uint24_t source`
	`uint32_t source`
	`uint64_t source`
		Integer mit Festkommaarithmetik, der in einen String umgewandelt werden soll.

	`int_fast8_t cp`	Source wird mit 10^{-cp} multipliziert
Rückgabewert:	`char *`	Zeiger auf den String target (vgl. Übergabewerte)
Empfohlene Verwendung:	`fixuint16str(target,source,cp);` `strRaglin(fixuint16str(str,source,cp),minlenght);`	
Bemerkungen:	keine	

1.1.6 strRalign

Prototyp:	`char *strRalign(char *str, uint_fast8_t minlenght);`	
Beschreibung:	Erweitert einen String um Leerzeichen. Die Leerzeichen werden vor den ursprünglichen Text gesetzt, so dass Zeichen in Strings nach rechts ausgerichtet werden können und der String eine fixe Länge hat. Ist der String zum Übergabezeitpunkt länger oder gleich lang wie der resultierende String, so wird der String nicht verändert.	
Übergabewerte:	`char *str`	Zeiger auf den String, der bearbeitet werden soll.
	`uint_fast8_t minlenght`	Minimale Länge des resultierenden Strings.
Rückgabewert:	`char *`	Zeiger auf den String str (vgl. Übergabewerte).

Empfohlene Verwendung:	`strRaglin(str,minlenght);`
	`strRaglin(int16str(str,source),minlenght);`
	`strRaglin(fixint16str(str,source,cp),minlenght);`	

Bemerkungen:	Der übergebene String darf nicht länger sein als UINT_FAST8_MAX (definiert in stdint.h), da ansonsten Fehlfunktionen auftreten können.	

1.2 Private Funktionen

Die privaten Funktionen sind auf ihren spezifischen Anwendungszweck innerhalb der Bibliothek abgestimmt. Daher sind diese Funktionen nicht besonders anwenderfreundlich, da sie eventuell keine Absicherung gegenüber unzulässigen Übergabewerten besitzen und keine Fehlercodes zurückgeben.
Falls diese Funktionen aus der Bibliothek exportiert und anderweitig verwendet werden, wird empfohlen, diese zu überarbeiten.

1.2.1 strswap

Prototyp:	`static void strswap(char *str, uint_fast8_t lenght);`	
Beschreibung:	Ändert die Reihenfolge aller Zeichen innerhalb eines Strings.	
Übergabewerte:	`char *str`	Zeiger auf den String, der bearbeitet werden soll.
	`uint_fast8_t lenght`	Länge des Strings exkl. Nullzeichen.
Rückgabewert:	`void`	Kein Rückgabewert.
Empfohlene Verwendung:	`strswap(str,strlen(str));` `/* strlen ist in string.h enthalten */`	
Bemerkungen:	Die effektive Stringlänge wird in der Funktion nicht überprüft. Ist der Übergabewert lenght zu gross, so können Speicherbereiche überschrieben werden, die nicht für den String reserviert sind.	

1.3 Öffentliche Makros

Es existieren keine öffentlichen Makros.

1.4 Private Makros

Es wird dringend empfohlen, die privaten Makros nicht direkt aufzurufen, sondern den sicheren Umweg über die Funktionen zu verwenden. Aus diesem Grund sind diese Makros nicht in der Headerdatei, sondern in der Sourcedatei enthalten und werden an dessen Ende wieder ungültig. Diese Makros werden hier nicht näher beschrieben, weitere Informationen können bei Bedarf direkt aus dem Quellcode entnommen werden.
Alle privaten Makros wurden nur aus einem einzigen Grund geschrieben: Da viele Funktionen den selben Code verwenden, sollen damit Doppelspurigkeiten vermieden werden und zukünftige Änderungen am Code vereinfacht werden, da sie nur an einer Stelle vorgenommen werden müssen. Leider leidet darunter die Lesbarkeit des Codes enorm.

1.5 Öffentliche globale Variablen

Es werden keine öffentlichen globalen Variablen verwendet.

1.6 Private globale Variablen

Es werden keine privaten globalen Variablen verwendet.

2 Versionsgeschichte

2.1 V0.0.0

Datum:	06.04.2013
Autor:	bestucki
Dateien:	Headerdateien: - intstr.h Sourcedateien: - intstr.c Sonstige Dateien: - Anwenderdokumentation_intstr_V00.pdf - _LICENCE.txt
Abhängigkeiten:	Headerdateien: - stdint.h
Lizenz:	GNU GENERAL PUBLIC LICENSE, Version 3, 29 June 2007 http://www.gnu.org/licenses
Änderungen:	keine (erste offizielle Version)
Bekannte Fehler:	In Funktion strRalign: - Bei der Ermittlung der Stringlänge wird das entsprechende Zeichen gegen 0 (Null) geprüft, richtig wäre jedoch '\0'. In den meisten Fällen wird dies keine Probleme bereiten, da 0 (Null) gleich '\0' ist. Alternativ kann die Sourcedatei manuell geändert oder auf die nächste offizielle Version gewartet werden.
Ankündigungen:	keine