HANSER

Leseprobe

Matthias Sturm

Mikrocontrollertechnik

Am Beispiel der MSP430-Familie

ISBN (Buch): 978-3-446-42231-5

ISBN (E-Book): 978-3-446-42964-2

Weitere Informationen oder Bestellungen unter

http://www.hanser-fachbuch.de/978-3-446-42231-5

sowie im Buchhandel.

2 Mikrorechentechnik-Grundlagen

Gleich zu Beginn wollen wir einige grundlegende Begriffe nennen. 1 **Bit** ist die kleinste Darstellungseinheit binärer Daten und kann die Werte 0 oder 1 annehmen. 4 Bit werden als **Nibble**, 8 Bit als **Byte** und 16 Bit als **Word** bezeichnet.

Auch ist es wichtig zu wissen, dass in der Mikrorechentechnik bei Aufzählungen oder Nummerierungen meistens mit der Zahl 0, und nicht mit der Zahl 1 begonnen wird.

2.1 Codes

Codes sind Vorschriften zur eindeutigen Zuordnung von Zeichenmengen. Ziffern und Zahlen, aber auch Buchstaben und Sonderzeichen werden in Rechnern codiert verarbeitet. Die gebräuchlichsten Codes wollen wir jetzt kennen lernen.

2.1.1 ASCII-Code

Zur Darstellung von Text hat sich sehr früh der ASCII-Code (American Standard Code for Information Interchange) etabliert. Die Codetabelle enthält neben Buchstaben und Ziffern auch Sonder- und Steuerzeichen. Wann immer Buchstaben, Zahlen oder Sonderzeichen auf einem Display ausgegeben werden sollen, ist der ASCII-Zeichensatz zu verwenden. Denn die meisten Hersteller alphanumerischer Displays haben sich auf die Verwendung des ASCII-Zeichensatzes verständigt. In ihm sind Textzeichen einem 7 Bit umfassenden Muster zugeordnet. Der Bitmusterumfang reicht vom Wert 0 bis 127. Da Rechner meist mit einer internen Bitbreite von 2^n aufgebaut sind, wurde der ASCII-Code auf 8 Bit erweitert, so dass weitere 128 Sonderzeichen (zum Beispiel Umlaute) aufgenommen worden sind. Die genaue Zuordnung der Bitmuster von 128 bis 255 muss man jedoch dem Datenblatt des Displays entnehmen.

Die genaue Zuordnung der ersten 128 Zeichen in der ASCII-Tabelle ist im Anhang F zu finden.

2.1.2 BCD-Code

Möchte man Ziffern in Mikrorechnern verarbeiten, so müssen diese in einen Binärcode umgewandelt werden. Eine mögliche Form, Ziffern binär darzustellen ist der BCD-Code (Binary Coded Decimal). Um die Ziffern 0 bis 9 darstellen zu können, werden zehn verschiedene Bitmuster benötigt. Sie sind in Tabelle 2.1 dargestellt.

Tabelle 2.1 BCD-Codierung

Dezimal-zahl	Binär-code	Dezimal-zahl	Binär-code	Dezimal-zahl	Binär-code	Dezimal-zahl	Binär-code
0	0000	4	0100	8	1000	ungenutzt	1100
1	0001	5	0101	9	1001	ungenutzt	1101
2	0010	6	0110	ungenutzt	1010	ungenutzt	1110
3	0011	7	0111	ungenutzt	1011	ungenutzt	1111

Da für eine Ziffer 4 Bit zur Codierung notwendig sind, kann ein Byte zwei BDC-codierte Ziffern tragen. Man spricht dann von *gepacktem BCD-Format*. Auffällig beim BCD-Code ist, dass von den 16 verschiedenen Bitmustern, die aus 4 Bit gebildet werden können, nur 10 genutzt werden, also ganze 62,5 %. Mit den meisten Zentraleinheiten ist es möglich, auf das BCD-Format einfache Addition und Subtraktion anzuwenden. Andere Zahlendarstellungen haben sich gegenüber dem BCD-Code in Mikrorechnern durchgesetzt. Vorteilhaft anzuwenden ist er jedoch bei der Zifferndarstellung, zumal es elektronische Bausteine gibt, die es erlauben 7-Segment-Anzeigen direkt im BCD-Format anzusteuern.

2.2 Darstellung von Zahlen in Mikrorechnern

Alle Zahlensysteme, mit denen wir umgehen, sind sogenannte Stellenwertsysteme. Dabei bestimmt nicht allein das Ziffernelement den Wert der Zahl, sondern auch die Position, an der die Ziffer steht.

$$Z = A \cdot x^n + B \cdot x^m + \ldots$$

Das dezimale Zahlensystem ist uns aus dem täglichen Umgang bekannt. Die Ziffern 0 bis 9 sowie die Potenzen zur Basis 10 bilden dieses Zahlensystem. Wir wenden es an, ohne wirklich über den Aufbau der Zahlen nachzudenken. Vielmehr haben wir ein Gefühl für Zahlen entwickelt. Es fällt nicht schwer, Zahlen aufsteigend zu sortieren oder ganzzahlige Vorgänger bzw. Nachfolger einer Zahl zu nennen. Diese Fähigkeit sollen wir auch für andere Zahlensysteme entwickeln.

2.2.1 Binäres Zahlensystem

Binäre Zahlen werden durch ein nachgestelltes kleines b gekennzeichnet. Innerhalb der binären Zahlendarstellung gibt es verschiedene Möglichkeiten, die verfügbare Bitbreite des Rechners zu nutzen. Die folgenden Erläuterungen beziehen sich auf einen 8 Bit breiten Rechenraum. Dadurch fällt es leichter, die Darlegungen zu verstehen.

2.2.1.1 Vorzeichenlose ganze Zahlen

Das Binärsystem kennt nur die Ziffern 0 und 1. Die Basis des Zahlensystems bildet die 2. Folgt man diesen Gegebenheiten, so erkennt man den darstellbaren Werteumfang eines Bytes. Es

sind 256 verschiedene Bitmuster darstellbar. Bei der angegebenen Wertigkeit von 2^0 bis 2^7 ergeben sich die Zahlen 0 bis 255 als der darstellbare Werteumfang. Die folgende Darstellung verdeutlicht die Wertzuordnung im Binärsystem.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	0	1	0	1	1	1

$$1001\,0111\,\text{binär} = 1\cdot 2^7 + 1\cdot 2^4 + 1\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0 = 151\,\text{dezimal}.$$

Das Bitmuster 0111 1011 entspricht folglich dem dezimalen Zahlenwert 123. Bei Binärzahlen, die 2 Byte umfassen, ergibt sich eine Darstellung von 0 (2^0) bis 65535 (2^{15}).

Wir wollen uns bereits an dieser Stelle angewöhnen, zwischen den einzelnen Nibbles (jeweils 4 Bit) eine kleine Lücke zu lassen, das erhöht die Übersichtlichkeit.

2.2.1.2 Vorzeichenlose gebrochene Zahlen, Festkommazahlen

Prinzipiell werden vorzeichenlose gebrochene Zahlen wie vorzeichenlose ganze Zahlen dargestellt. Lediglich die Exponenten sind andere. Im folgenden Beispiel sind die Wertigkeiten 2^{-7} bis 2^0 festgelegt. Der darstellbare Werteumfang reicht hier von 0 bis 1,9921875 bei einer kleinsten Schrittweite von 0,0078125.

2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
1	0	0	1	0	1	1	1

$$\begin{aligned} 1001\,0111\,\text{binär} &= 1\cdot 2^0 + 1\cdot 2^{-3} + 1\cdot 2^{-5} + 1\cdot 2^{-6} + 1\cdot 2^{-7} \\ &= 1{,}1796875\,\text{dezimal}. \end{aligned}$$

2.2.1.3 Vorzeichenbehaftete ganze Zahlen

Vorzeichen-Betrags-Darstellung

Es gibt verschiedene Ansätze, vorzeichenbehaftete Zahlen in Mikrorechnern darzustellen. Am einfachsten erscheint die Möglichkeit, das höchstwertige Bit, das MSB (Most Significant Bit), als Vorzeichenbit einzusetzen. Gesetztes Bit 7 bedeutet negative Zahl, gelöschtes Bit 7 bedeutet positive Zahl. Dann würde das Bitmuster aus den vorangegangenen Beispielen 1001 0111 der Zahl −23 entsprechen. Ein denkbar einfaches Prinzip – jedoch mit einem gravierenden Nachteil: Es gibt zwei Darstellungen der Zahl 0, nämlich +0 und −0. Oder anders dargestellt: Die Bitmuster 0000 0000 und 1000 0000 besitzen den gleichen Zahlenwert. Dass zwei unterschiedliche Bitmuster das Gleiche bedeuten, ist dem Rechner nur schwer klar zu machen. Auch gibt es gravierende Probleme bei arithmetischen Operationen, wie der Addition und Subtraktion.

Einerkomplementdarstellung

Man könnte jetzt auf die Idee kommen, durch die Komplementbildung einer Zahl (gemeint ist die bitweise Umkehr jedes einzelnen Bitzustandes der Zahl) das Vorzeichen zu ändern. Aus +1 wird −1 durch das Komplement des Bitmusters 0000 0001 zu 1111 1110. So weit, so gut. Aber auch hier gibt es zwei unterschiedliche Darstellungen für die Ziffer 0 (00000000 =

+0/11111111 = −0). Auch bei dieser Zahlendarstellung sind Fehler bei der Berechnung nicht auszuschließen.

Zweierkomplementdarstellung

Einzig die Zweierkomplementdarstellung vorzeichenbehafteter ganzer Zahlen führt in Mikrorechnern zum richtigen Ergebnis. (Diese Aussage gilt nicht generell. Sie werden bei hochintegrierten Prozessoren weitere Zahlendarstellungen antreffen, bei denen weitaus kompliziertere Konstrukte dank eines angepassten Rechenwerkes mühelos verarbeitet werden.) Die Zweierkomplementdarstellung bildet aus einer positiven Zahl eine negative Zahl gleichen Betrages, indem zuerst das Einerkomplement gebildet und anschließend die Zahl 1 addiert wird. Aus +1 wird −1 durch das Einerkomplement von 0000 0001 → 1111 1110 und anschließende Addition von 1: 1111 1110 + 0000 0001 = 1111 1111. Im Zweierkomplementsystem gibt es nur eine Darstellung der Zahl 0. Allerdings sind der positive und negative Zahlenraum nicht mehr symmetrisch, oder anders gesagt: Positive und negative Zahlen sind nicht mehr gleich verteilt. Innerhalb eines Bytes kann man die Ziffern +127 … 0 … − 128 darstellen.

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	0	1	0	1	1	1

$$1001\,0111\,\text{binär} = 1\cdot(-2^7) + 1\cdot 2^4 + 1\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0 = -105\,\text{dezimal}.$$

Zweierkomplementdarstellung gebrochener Zahlen

Verändert man die Exponenten der Potenzen, die den jeweiligen Stellen zugeordnet sind, so sind mühelos gebrochene Zahlen darstellbar. Bei einer Exponentenfolge von 0 bis −7 entspricht die Zahl 1111 1110 somit −0,015625. Der Wertebereich erstreckt sich von −1 über 0 bis fast +1.

-2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
1	0	0	1	0	1	1	1

$$\begin{aligned}1001\,0111\,\text{binär} &= 1\cdot(-2^0) + 1\cdot 2^{-3} + 1\cdot 2^{-5} + 1\cdot 2^{-6} + 1\cdot 2^{-7}\\ &= -0{,}8203125\,\text{dezimal}.\end{aligned}$$

2.2.2 Informationsgehalt eines Bytes

Betrachten wir rückblickend noch einmal den Informationsgehalt eines Bytes, so stellen wir zugleich Erschreckendes und Erfreuliches fest. Das Bitmuster in einem Byte kann sein:

- eine ganze Zahl zwischen 0 und +255 (Dualcode),
- eine ganze Zahl zwischen −128 und +127 (Zweierkomplement),
- eine Zahl zwischen 00 und 99 (im gepackten BCD-Format),
- ein Buchstabe, eine Ziffer oder ein Sonder- bzw. Steuerzeichen (ASCII-Code),
- ein 1-Byte-Befehlswort (siehe Kapitel 1 Ein-Bit-Rechner),
- oder was immer der Programmierer festgelegt hat!

Auf die Frage, was ein Bitmuster bedeutet, gibt es folglich keine eindeutige Antwort. Was die Sache noch weiter verkompliziert, ist der Umstand, dass das unveränderte Bitmuster seine Bedeutung im Laufe des Programms ändern kann.

Diese Freiheit bedrückt den Einsteiger zu Anfang. Mit zunehmender Erfahrung aber sind es gerade diese Freiräume, die der Kreativität des Programmierers keine Grenzen setzen.

Der Programmierer ist Chef im System, was er entscheidet und festlegt, ist Gesetz.

2.2.3 Hexadezimales Zahlensystem

Zugegeben, die Darstellung von Zahlen und Zeichen in Binärdarstellung ist etwas unübersichtlich. Daran ändert auch der Umstand nichts, dass wir bei der Darstellung der dualen Bitmuster die Nibble separieren. Um die Übersichtlichkeit zu steigern – und wohl auch ein wenig aus Bequemlichkeit – wurden weitere Zahlensysteme eingeführt. Für die Mikrorechentechnik ist dabei das Hexadezimalsystem von besonderer Bedeutung. Mit ihm gelingt es, jedem Nibble ein Ziffernelement zuzuordnen, so dass man für die Darstellung eines Bytes nur zwei Stellen benötigt. Mathematisch ist das Hexadezimalsystem ein Stellenwertsystem zur Basis 16. Da man aber nicht 16 verschiedene Ziffernelemente verfügbar hatte, entschloss man sich, die Buchstaben A bis F als Ziffernelemente zu verwenden. Die folgende Tabelle 2.2 zeigt eine Gegenüberstellung des Dezimal- und Hexadezimalsystems.

Tabelle 2.2 Hexadezimaldarstellung

Dezimal-zahl	Hexa-dezimal-code	Dezimal-zahl	Hexa-dezimal-code	Dezimal-zahl	Hexa-dezimal-code	Dezimal-zahl	Hexa-dezimal-code
0	0	4	4	8	8	12	C
1	1	5	5	9	9	13	D
2	2	6	6	10	A	14	E
3	3	7	7	11	B	15	F

Anfangs fällt es schwer zu verstehen, dass bei der hexadezimalen Zahlendarstellung die Zeichen A … F gleichwertige Ziffernzeichen wie 1 … 9 sind. Bei einer intensiven Beschäftigung mit Mikrorechnern gewöhnt man sich jedoch schnell an diese Darstellungsform.

Folgender *Hinweis* ist außerdem sehr wichtig: Die Umwandlung von Dual- oder Hexadezimalzahlen in Dezimalzahlen und umgekehrt ist selten notwendig. Die Konvertierung zwischen Dual- und Hexadezimalzahlen aber ist bei der Arbeit mit Mikrorechnern ständig zu bewältigen. Man sollte also bei einer Hexadezimalzahl immer auch die duale Entsprechung vor sich sehen, da oft einzelne Bits für das Ein- und Ausschalten von Aktoren verwendet werden. Zum Glück ist das Konvertieren zwischen hexadezimal und dual dargestellten Bitmustern denkbar einfach. Jeweils 4 Bit (ein Nibble) werden zu einer hexadezimalen Zahl zusammengefasst. Umgekehrt repräsentiert jede hexadezimale Ziffer 4 Bit. Man muss also nicht den Wert der Zahl kennen, sondern lediglich Nibble für Nibble die Konvertierung voran treiben.

2.2.4 Zahlendarstellung in 16-Bit-Systemen

Die MSP430-Bausteine, anhand derer wir in diesem Buch die Funktion der Mikrocontroller kennen lernen werden, verfügen über ein natives Datenformat von 16 Bit. Folglich ist auch der Rechnerkern 16 Bit breit aufgebaut. Durch die schaltungstechnische Umsetzung in Rechnern ergeben sich geschlossene Zahlenräume. Die Rechner bestehen zum großen Teil aus Flipflops und Zählern, die nach einer bestimmten Anzahl von Takten wieder ihren Ausgangszustand annehmen, daher die geschlossenen Zahlenräume. Bei der Nutzung des 16-Bit-Zahlenraumes mit vorzeichenloser Zahlendarstellung ergibt sich die Darstellung nach Bild 2.1.

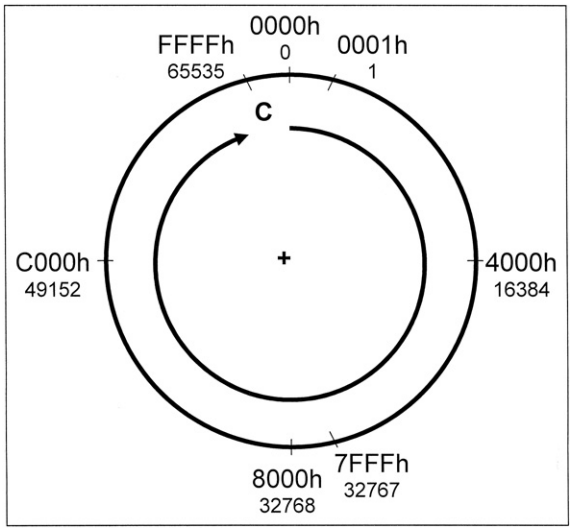

Bild 2.1 Vorzeichenloses 16-Bit-Hexadezimalsystem

Bei der Verwendung der Zweierkomplementdarstellung sieht das anders aus (Bild 2.2).

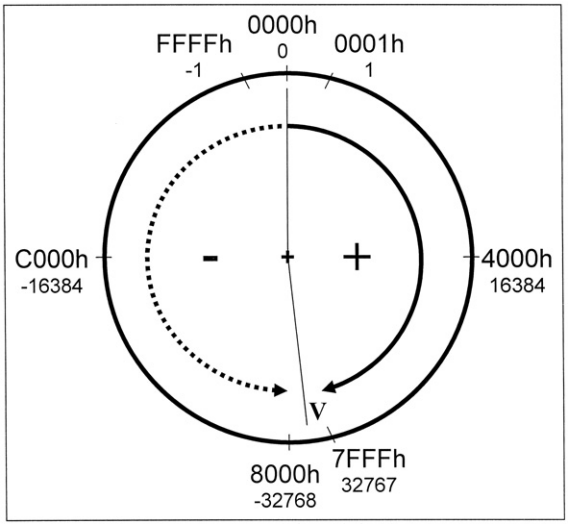

Bild 2.2 Zweierkomplementdarstellung im 16-Bit-Hexadezimalsystem

Nun sind Mikrorechner bei ihrer Anwendung nicht auf den Wertevorrat ihrer Zahlenräume beschränkt. Vielmehr sind dies die Bereiche, innerhalb derer sich der Rechnerkern mit einer einzigen Operation bewegen kann. Sollen größere Zahlen verarbeitet werden, so ist der Status des Rechenwerkes hinzuzuziehen. Dabei helfen Flags, die im Anschluss vorgestellt werden.

2.3 Statusbits

Der Wertebereich von Zahlen in Mikrorechnern ist auf Grund der verfügbaren Bitbreite begrenzt. Dies erkennt man leicht an den Darstellungen in den Bildern 2.1 und 2.2. Eine Besonderheit aller Rechner ist aber, dass der Wertebereich ein geschlossenes System bildet. Der Grund liegt in der Hardware der Rechnerkerne. Eingesetzte Zählerschaltungen beenden ihre Aktivitäten nicht an den Grenzen des Wertebereichs. Sie laufen über bzw. unter und beginnen wieder an der entgegengesetzten Grenze des Wertebereichs.

Da Berechnungen aber nur innerhalb des verfügbaren Wertebereichs richtig ausgeführt werden, ist es notwendig zu wissen, ob dieser verlassen wurde. Am Rechenergebnis selbst kann man das Überschreiten des Wertebereichs nicht unmittelbar erkennen.

Hier kommen die Zustandsanzeiger oder auch Flags zum Einsatz. Wir haben ein Flag bereits im Kapitel 1 über den Ein-Bit-Rechner kennen gelernt. Vier Flags wollen wir uns nun näher anschauen.

Voranzustellen ist der wichtige Hinweis, dass sich Flags stets nur im Ergebnis von logischen oder arithmetischen Operationen einstellen (von ganz wenigen Ausnahmen abgesehen). Die Befehlsliste im Anhang C1 gibt Aufschluss darüber, welche Befehle Flags manipulieren. Flags dienen der Beeinflussung des Programmlaufes oder einfacher: Flags können zur bedingten Programmverzweigung eingesetzt werden.

2.3.1 Z-Flag

Das **Nullflag** (zero flag oder Z-Flag) ist der Indikator für den Ergebniswert null. Hat sich der Wert null als Ergebnis eingestellt, so ist das Z-Flag gesetzt, d. h. auf 1 geschaltet. In allen anderen Fällen ist es gelöscht, d. h. logisch 0.

Möchte man beispielsweise einen Rückwärtszählvorgang an der Stelle null beenden, so ist es nicht notwendig, nach jedem Zählschritt das Ergebnis anzusehen. Es reicht aus, das Z-Flag auszuwerten.

2.3.2 N-Flag

Interpretiert man das aktuell im Rechenwerk bearbeitete Bitmuster als vorzeichenbehaftete Zahl, so zeigt das gesetzte **N-Flag** (negative flag) eine negative Zahl an und das gelöschte N-Flag folglich eine positive Zahl.

Um zu erfahren, ob eine Berechnung eine negative Zahl ergeben hat, muss man nicht das Ergebnis betrachten. Die Auswertung des N-Flags reicht völlig aus.

Aus den vorangegangenen Erläuterungen wissen wir, dass bei Zweierkomplementzahlen das MSB (Most Significant Bit, das ganz links stehende Bit) kennzeichnend für eine negative Zahl ist. Man kann also sagen, dass das N-Flag nach der Anwendung bestimmter Befehle eine Kopie des MSB trägt.

Was aber, wenn man vorzeichenlose Zahlen bearbeitet, bei denen das MSB gesetzt ist? Auch in diesen Fällen wird das N-Flag beeinflusst, man muss es ja nicht beachten oder auswerten.

2.3.3 C-Flag

C steht für Carry oder Übertrag. Immer wenn sich bei Additionen oder Subtraktionen ein Übertrag ergibt, wird dieser im C-Flag registriert.

Das gesetzte **C-Flag** signalisiert einen Übertrag, das gelöschte C-Flag entsprechend keinen Übertrag. Zusätzlich wird das Carry Flag bei einigen logischen Befehlen als Zwischenspeicher genutzt, doch dazu später.

Ein Übertrag kann bei der Addition bzw. Subtraktion vorzeichenloser Zahlen entstehen. Dies ist der Fall, wenn der Wertebereich vom maximalen Zahlenwert in Richtung Null überläuft oder wenn bei einer Subtraktion der minimal darstellbare Zahlenwert unterlaufen wird.

2.3.4 V-Flag

Das Überlaufflag (overflow flag oder **V-Flag**) signalisiert das Verlassen des darstellbaren Wertebereichs vorzeichenbehafteter Zahlen. Dies tritt bei der Kombination folgender Zahlen und Operationen ein:

positiv	+	positiv	=	negativ
negativ	+	negativ	=	positiv
positiv	−	negativ	=	negativ
negativ	−	positiv	=	positiv.

Woher aber weiß die CPU, um was für eine Zahlendarstellung es sich bei dem aktuellen Bitmuster im Rechenwerk handelt? Denn nur so könnte doch das ‚richtige' Flag beeinflusst werden.

Die einfache Antwort ist: Die CPU hat keine Ahnung! Sie beeinflusst die Flags für alle in Frage kommenden Zahlenformate. Der Anwender ist gefordert, das für das aktuelle Zahlenformat richtige Flag zur Programmflusssteuerung auszuwählen. Ausgewählt wird letztlich nicht ein bestimmtes Flag, sondern ein Verzweigungsbefehl, der in Abhängigkeit des gewünschten Flagzustandes eine Verzweigung ausführt – oder eben nicht.