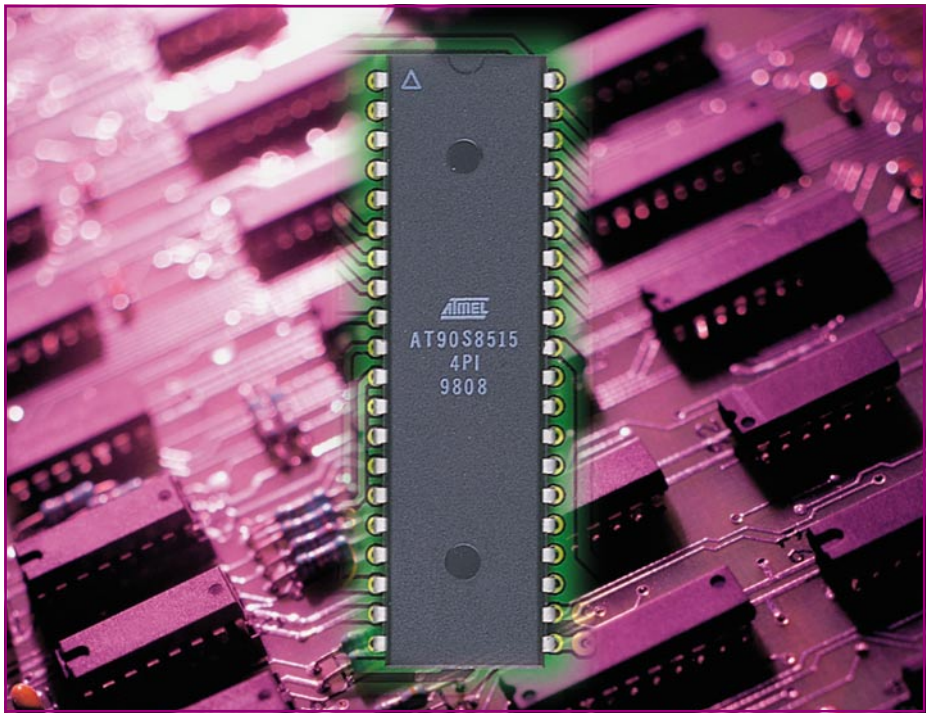

AVR-Grundlagen Teil 2

Die AVR-Mikrocontroller erfreuen sich stetig wachsender Beliebtheit unter Schaltungsentwicklern sowohl im professionellen als auch im privaten Bereich. Sie zeichnen sich durch eine hohe Verarbeitungsgeschwindigkeit und eine Vielzahl verschiedener Typen aus, sodass man quasi zu jedem Projekt über den passenden Controller verfügen kann. In dieser Artikelserie wird die Controller-Baureihe vorgestellt und anhand von Anwendungsbeispielen erläutert. Nachdem wir im ersten Teil den Aufbau und die grundlegende Beschaltung der Controller besprochen haben, erfolgt nun die Beschreibung des Befehlssatzes, die Vorstellung der Register mit ihren speziellen Funktionen, z. B. zur Timersteuerung, dem Zugriff auf das interne EEPROM usw. Schließlich beschreiben wir den Grundaufbau eines Programms im AVR-Assembler.

Der Befehlssatz

Der Befehlssatz der AVR-Mikrocontroller umfasst 89 bis 128 Befehle, abhängig vom verwendeten Controllertyp. Welche Befehle für den einzelnen Mikrocontroller verfügbar sind, entnimmt man dem jeweiligen Datenblatt, das unter [1] verfügbar ist. Um sich einen ersten Überblick zu verschaffen, sind die Befehle in Gruppen aufgeteilt: arithmetische und logische Befehle, Bit- und Bit-Test-Funktionen, Datentransfer und Sprungbefehle.

Bevor wir tiefer in den Befehlssatz einsteigen, müssen die notwendigen Register und Operanden betrachtet werden.

Register

Der Registerbereich umfasst 32 Register zu je 8 Bit, welche über die Bezeichner „R0" bis „R31" ansprechbar sind.

Neben der Registerbank gibt es den Bereich des Datenspeichers, den man über zwei verschiedene Arten adressieren kann. Bei der direkten Adressierung wird die Speicherstelle unmittelbar über die Adresse oder den zugehörigen Bezeichner angesprochen, d. h. der Zahlenwert der Adresse wird als Operand des entsprechenden Befehls angegeben. Andere Befehle unterstützen die indirekte Adressierung, d. h., dass auf Daten im SRAM nicht über die konstante Adresse, die fest im Programm verankert ist, zugegriffen wird, sondern die Speicheradresse wird während der Laufzeit in einem bestimmten Bereich der Registerbank abgespeichert und vom Befehl als Adresse interpretiert. Eine solche Registerbankvariable bezeichnet man auch als „Zeiger", da der enthaltene Wert auf einen Bereich im Speicher zeigt. Diese Zeiger sind beim AVR-Assembler mit X, Y und Z deklariert und aus jeweils zwei Registern zusammengesetzt (X=R27:R26, Y=R29:R28, Z=R31:R30). Sie haben eine Breite von 16 Bit und können somit einen Speicherbereich von bis zu 64 KB adressieren.

Ein weiteres Register ist das Statusregister, welches die wichtigsten Flags enthält.

Flags

Ein Flag (Flagge) ist ein Bit, das das Auftreten bestimmter Ereignisse oder Zustände kenntlich macht und nur die Werte Null oder Eins annehmen kann.

Das Carry-Flag wird immer dann gesetzt, wenn es einen Überlauf bei einer Operation gegeben hat. Addiert man mit 8 Bit breiten Registern (max. Zahlenwert 255) die Werte 230 und 56, dann gibt es einen Überlauf, da das Zielregister den Zahlenwert von 286 nicht aufnehmen kann. Stattdessen wird das „Carry-Flag" (C) zur Erkennung des Überlaufs in die nächsthöhere Stelle, welche einen Wert von 256 hat, gesetzt. Das Zielregister enthält danach den Wert 30. Es geht keine Information verloren.

Ein weiteres wichtiges Flag ist das „Zero-Flag" (Z). Es wird gesetzt, wenn das Ergebnis einer mathematischen Operation gleich Null ist. Ein gesetztes Zero-Flag nach einer Subtraktion bedeutet, dass die beiden Operanden den gleichen Wert hatten.

Das „Negativ-Flag" (N) kennzeichnet ein negatives Ergebnis, falls man vorzeichenbehaftete Operationen durchführt. Ansonsten ist das Negativ-Flag immer identisch mit der höchstwertigsten Stelle des Ergebnisses.

Befehlsübersichten

Eine Übersicht über diese und weitere Flags befindet sich in Tabelle 1, ebenso wie die in der Befehlsbeschreibung verwendeten Abkürzungen.

Die Tabellen 2 bis 5 enthalten eine Kurzübersicht aller von den AVR-Mikrocontrollern unterstützten Befehle. Eine ausführliche Beschreibung findet man unter [2] auf der Atmel-Homepage.

In diesen Tabellen gibt es jeweils drei Spalten mit den wichtigsten Informationen. Die erste Spalte zeigt den Befehl, in der zweiten stehen die notwendigen Operanden, gefolgt von der Kurzbeschreibung.

Tabelle 1: Bezeichner zur Beschreibung des Befehlssatzes

Statusregister (SREG)		Register und Operanden	
SREG:	Statusregister	Rd:	Ziel- und Quellregister
C:	Carry-Flag	Rr:	Quellregister
Z:	Zero Flag	R:	Ergebnis nach Befehls-ausführung
N:	Negative-Flag	K:	Konstante Daten
V:	Flag zur Erkennung eines Überlaufs beim Zweier-Komplement	k:	Konstante Adresse (Direkte Adressierung)
S:	Sign-Flag (Vorzeichen)	b:	Bit in Registerbank oder im I/O-Bereich
H:	Half-Carry-Flag	s:	Bit im Statusregister
T:	Transfer-Bit (wird von BLD- und BST-Befehlen verwendet)	X,Y,Z:	Register zur indirekten Adressierung
		A:	I/O-Adresse
		PC:	Befehlszähler
I:	Globales Interrupt-Enable-Flag		

In der nächsten Spalte sind alle Flags aufgeführt, die von diesem Befehl beeinflusst werden. Abschließend findet man noch die benötigten Taktzyklen, die zur Ausführung der Funktion notwendig sind.

Tabelle 2 zeigt die Arithmetik- und Logikbefehle, mit denen mathematische Funktionen realisierbar sind. Zu den Grundfunktionen gehören hier die Addition, die Subtraktion sowie die Multiplikation zwischen zwei Registern oder einem Register und einer Konstanten. Des Weiteren sind Befehle zum gezielten Verändern eines Registers enthalten (Inkrementieren, Dekrementieren, Löschen, Setzen usw.). Andere mathematische Funktionen (z. B. Division) lassen sich durch kleine Routinen mit den vorhandenen Befehlen einfach realisieren.

Um die in einem Programmablauf notwendigen Verzweigungen ausführen zu können, verfügen die AVR-Mikrocontroller über vielfältige Sprunganweisungen, wie man sie in Tabelle 3 zusammengefasst findet.

Am Anfang der Tabelle befinden sich die Sprunganweisungen, die beim Aufruf an die angegebene Adresse im Programmspeicher springen (RJMP, IJMP, EIJMP, JMP). Anschließend sind die Befehle zur Behandlung von Unterfunktionen (Subroutine) und Interrupt-Service-Routinen angegeben. Ein Funktionsaufruf erfolgt über einen „CALL"-Befehl (RCALL, ICALL, EICALL, CALL). Da bei einem solchen Befehl die spätere Rücksprungadresse im Stack gespeichert wird, muss der Rücksprung zur aufrufenden Routine mit dem „RET"-Befehl erfolgen, der die Adresse wieder aus dem Stack holt und damit das Programm an der korrekten Stelle fortführt.

Interrupt-Service-Routinen, die beim Auftreten einer Unterbrechungsanforderung automatisch aufgerufen werden, müssen mit dem „RETI"-Befehl verlassen werden, da der Aufruf und Rücksprung andere Prozessoraktionen als bei einer normalen Unterroutine auslöst.

Die bisher noch nicht besprochenen Befehle dienen zur Ausführung bedingter Verzweigungen. Der Befehl „CPSE" vergleicht die beiden Operanden und überspringt den folgenden Befehl, falls der Inhalt dieser Register gleich ist. Alle weiteren Befehle für Vergleiche (CP, CPC, CPI) führen keinen direkten Sprung aus, falls bestimmte

Tabelle 2: Arithmetik- und Logikbefehle

Mnemonics	Operands	Description
ADD	Rd, Rr	Add without Carry
ADC	Rd, Rr	Add with Carry
ADIW	Rd, K	Add Immediate to Word
SUB	Rd, Rr	Subtract without Carry
SUBI	Rd, K	Subtract Immediate
SBC	Rd, Rr	Subtract with Carry
SBCI	Rd, K	Subtract Immediate with Carry
SBIW	Rd, K	Subtract Immediate from Word
AND	Rd, Rr	Logical AND
ANDI	Rd, K	Logical AND with Immediate
OR	Rd, Rr	Logical OR
ORI	Rd, K	Logical OR with Immediate
EOR	Rd, Rr	Exclusive OR
COM	Rd	One's Complement
NEG	Rd	Two's Complement
SBR	Rd,K	Set Bit(s) in Register
CBR	Rd,K	Clear Bit(s) in Register
INC	Rd	Increment
DEC	Rd	Decrement
TST	Rd	Test for Zero or Minus
CLR	Rd	Clear Register
SER	Rd	Set Register
MUL	Rd,Rr	Multiply Unsigned
MULS	Rd,Rr	Multiply Signed
MULSU	Rd,Rr	Multiply Signed with Unsigned
FMUL	Rd,Rr	Fractional Multiply Unsigned
FMULS	Rd,Rr	Fractional Multiply Signed
FMULSU	Rd,Rr	Fractional Multiply Signed with Unsigned

Tabelle 3: Sprung-Anweisungen

Mnemonics	Operands	Description
RJMP	k	Relative Jump
IJMP		Indirect Jump to (Z)
EIJMP		Extended Indirect Jump to (Z) .
JMP	k	Jump
RCALL	k	Relative Call Subroutine
ICALL		Indirect Call to (Z)
EICALL		Extended Indirect Call to (Z)
CALL	k	Call Subroutine
RET		Subroutine Return
RETI		Interrupt Return
CPSE	Rd,Rr	Compare, Skip if Equal
CP	Rd,Rr	Compare
CPC	Rd,Rr	Compare with Carry
CPI	Rd,K	Compare with Immediate
SBRC	Rr, b	Skip if Bit in Register Cleared
SBRS	Rr, b	Skip if Bit in Register Set
SBIC	A, b	Skip if Bit in I/O Register Cleared
SBIS	A, b	Skip if Bit in I/O Register Set
BRBS	s, k	Branch if Status Flag Set
BRBC	s, k	Branch if Status Flag Cleared
BREQ	k	Branch if Equal
BRNE	k	Branch if Not Equal
BRCS	k	Branch if Carry Set
BRCC	k	Branch if Carry Cleared
BRSH	k	Branch if Same or Higher
BRLO	k	Branch if Lower
BRMI	k	Branch if Minus
BRPL	k	Branch if Plus
BRGE	k	Branch if Greater or Equal, Signed
BRLT	k	Branch if Less Than, Signed
BRHS	k	Branch if Half Carry Flag Set
BRHC	k	Branch if Half Carry Flag Cleared
BRTS	k	Branch if T Flag Set
BRTC	k	Branch if T Flag Cleared
BRVS	k	Branch if Overflow Flag is Set
BRVC	k	Branch if Overflow Flag is Cleared
BRIE	k	Branch if Interrupt Enabled
BRID	k	Branch if Interrupt Disabled

Tabelle 4: Befehle zum Datentransfer

Mnemonics	Operands	Description
MOV	Rd, Rr	Copy Register
MOVW	Rd, Rr	Copy Register Pair
LDI	Rd, K	Load Immediate
LDS	Rd, k	Load Direct from data space
LD	Rd, X	Load Indirect
LD	Rd, X+	Load Indirect and Post-Increment
LD	Rd, -X	Load Indirect and Pre-Decrement
LD	Rd, Y	Load Indirect
LD	Rd, Y+	Load Indirect and Post-Increment
LD	Rd, -Y	Load Indirect and Pre-Decrement
LDD	Rd,Y+q	Load Indirect with Displacement
LD	Rd, Z	Load Indirect
LD	Rd, Z+	Load Indirect and Post-Increment
LD	Rd, -Z	Load Indirect and Pre-Decrement
LDD	Rd, Z+q	Load Indirect with Displacement
STS	k, Rr	Store Direct to data space
ST	X, Rr	Store Indirect
ST	X+, Rr	Store Indirect and Post-Increment
ST	-X, Rr	Store Indirect and Pre-Decrement
ST	Y, Rr	Store Indirect
ST	Y+, Rr	Store Indirect and Post-Increment
ST	-Y, Rr	Store Indirect and Pre-Decrement
STD	Y+q,Rr	Store Indirect with Displacement
ST	Z, Rr	Store Indirect
ST	Z+, Rr	Store Indirect and Post-Increment
ST	-Z, Rr	Store Indirect and Pre-Decrement
STD	Z+q,Rr	Store Indirect with Displacement
LPM		Load Program Memory
LPM	Rd, Z	Load Program Memory
LPM	Rd, Z+	Load Program Memory and Post-Increment
ELPM		Extended Load Program Memory
ELPM	Rd, Z	Extended Load Program Memory
ELPM	Rd, Z+	Extended Load Program Memory and Post-Increment
SPM		Store Program Memory
ESPM		Extended Store Program Memory
IN	Rd, A	In From I/O Location
OUT	A, Rr	Out To I/O Location I/O
PUSH	Rr	Push Register on Stack
POP	Rd	Pop Register from Stack

Tabelle 5: Bit- und Bit-Test-Befehle

Mnemonics	Operands	Description
LSL	Rd	Logical Shift Left
LSR	Rd	Logical Shift Right
ROL	Rd	Rotate Left Through Carry
ROR	Rd	Rotate Right Through Carry
ASR	Rd	Arithmetic Shift Right
SWAP	Rd	Swap Nibbles
BSET	s	Flag Set
BCLR	s	Flag Clear
SBI	A, b	Set Bit in I/O Register
CBI	A, b	Clear Bit in I/O Register
BST	Rr, b	Bit Store from Register to T
BLD	Rd, b	Bit load from T to Register
SEC		Set Carry
CLC		Clear Carry
SEN		Set Negative Flag
CLN		Clear Negative Flag
SEZ		Set Zero Flag
CLZ		Clear Zero Flag
SEI		Global Interrupt Enable
CLI		Global Interrupt Disable
SES		Set Signed Test Flag
CLS		Clear Signed Test Flag
SEV		Set Two's Complement Overflow
CLV		Clear Two's Complement Overflow
SET		Set T in SREG
CLT		Clear T in SREG
SEH		Set Half Carry Flag in SREG
CLH		Clear Half Carry Flag in SREG
NOP		No Operation
SLEEP		Sleep
WDR		Watchdog Reset

Internet:

[1] Datenblätter:
http://www.atmel.com/atmel/products/prod200.htm

[2] Befehlssatz der AVR-Controller im PDF-Format:
http://www.atmel.com/atmel/acrobat/doc0856.pdf

Bedingungen erfüllt sind, sondern führen intern nur eine Subtraktion durch, welche die Flags im Statusregister entsprechend setzt oder zurücksetzt. Ein nachfolgender Befehl zum Verzweigen (BRBS, BRBC, BREQ usw.) wertet diese Flags aus und führt bei positivem Ergebnis den Sprung an die angegebene Stelle des Programms aus.

Die meisten Befehle der AVR-Controller arbeiten nur mit der Registerbank und können nicht direkt auf den Datenspeicher zugreifen. Deshalb ist es notwendig, die Daten aus dem Datenspeicher im SRAM auf die Registerbank zu übertragen, sodass die entsprechenden Operationen durchgeführt werden können. In Tabelle 4 sind die Befehle zum Datentransfer zusammengefasst.

Der abschließende Teil der Befehle ist in Tabelle 5 zu sehen. Alle aufgeführten Anweisungen dienen zum Manipulieren oder zum Testen einzelner Bits.

Special-Function-Register (SFR)

Neben den 32 Arbeitsregistern der Registerbank, die für allgemeine Programmabläufe nutzbar sind, gibt es auch die so genannten „Special-Function-Register" oder kurz „SFR", denen, wie der Name schon sagt, spezielle Funktionen zugewiesen sind. Über diese SFR erfolgt die Steuerung und der Betrieb der im Mikrocontroller integrierten Peripherie (Timer/Counter, EEPROM, UART usw.). Diese Register sind also eng mit der Hardware verbunden und können somit zwischen den einzelnen Controllertypen variieren. Die genauen Eigenschaften und Bedeutungen der einzelnen Register zum passenden Mikrocontroller sind in den entsprechenden Datenblättern [2] aufgeführt.

Der Programmaufbau

Jeder AVR-Mikrocontrollertyp hat unterschiedliche Speichergrenzwerte, SFR usw., die in einer Datei zusammengefasst sind. In diesen Include-Dateien werden die in den Datenblättern angegebenen Bezeichnungen der Register und Bits mit den entsprechenden Adressen verknüpft. Zu Beginn eines jeden Programmes muss deshalb die controllerspezifische Datei eingebunden werden.

Das Programm startet nach einem Reset immer an der Adresse 0 im Programmspeicher, sodass dort die entsprechende Verzweigung zum eigentlichen Programmstart erfolgen muss.

Auf eine Unterbrechungsanforderung (Interrupt) erfolgt, bei eingeschaltetem Interrupt, der automatische Sprung an die festgelegte Adresse im Programmspeicher. Die Einsprungadressen der verschiedenen Interruptquellen starten im Programmspeicher ab 1. Das bedeutet, dass an diesen Stellen jeweils Verzweigungen zu den entsprechenden Interrupt-Service-Routinen erfolgen müssen, in denen die Bearbeitung der Unterbrechungsanforderung erfolgt.

Das Programm sollte nach einem Reset immer mit der Initialisierung der Special-Function-Register sowie der Variablen beginnen, damit das nachfolgende Programm beim Start immer die richtigen Bedingungen vorfindet.

Damit ist der zweite Teil der Artikelserie „AVR-Grundlagen" abgeschlossen. Im nächsten „ELVjournal" wird ein erstes Projekt anhand eines kleinen Beispielprogrammes implementiert, wobei wir die Entwicklungsumgebung für diese Mikrocontroller, das „AVR-Studio", nutzen.