ARM Assembler-Befehlssatz (Auszug)

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV {S}	Rd, Rn	Move Register	Rd:=Rn	{N,Z,C}	1
MVN {S}	Rd, Rn	Move Not (Bitweise)	Rd:=~Rn	{N,Z,C}	1
MOV	Rd, #imm16	Move Immediate	Rd[15:0]:=imm16		1
MOVT	Rd, #imm16	Move Top Immediate	Rd[31:16]:=imm16		1
ADD {S}	Rd,Rn, op2	Add	Rd:=Rn+op2	{N,Z,C,V}	1
ADC {S}	Rd,Rn, op2	Add with Carry	Rd:=Rn+op2+C	{N,Z,C,V}	1
ADD	Rd,Rn, #imm12	Add	Rd:=Rn+imm12		1
SUB {S}	Rd,Rn, op2	Subtract	Rd:=Rn-op2	{N,Z,C,V}	1
SBC {S}	Rd,Rn, op2	Subtract with borrow	Rd:=Rn-op2+C	{N,Z,C,V}	1
SUB	Rd,Rn, #imm12	Subtract	Rd:=Rn-imm12	{N,Z,C,V}	1
MUL {S}	Rd,Rn,Rm	Multiply	Rd:=Rn*Rm	{N,Z}	1
UDIV	Rd,Rn,Rm	Unsigned Divide	Rd:=Rn/Rm		2..12
CMP	Rn, op2	Compare	Rn-op2	N,Z,C,V	1
AND {S}	Rd,Rn, op2	And (Bitweise)	Rd:=Rn&op2	{N,Z,C}	1
ORR {S}	Rd,Rn, op2	Or (Bitweise)	Rd:=Rn|op2	{N,Z,C}	1
EOR {S}	Rd,Rn, op2	Exor (Bitweise)	Rd:=Rn^op2	{N,Z,C}	1
BIC	Rd,Rn, op2	And Not (Bit clear)	Rd:=Rn&~op2	N,Z,C	1
TST	Rn, op2	Test (Bitweise)	Rn&op2	N,Z,C	1
LSL {S}	Rd,Rn,Rs	Logical shift left	Rd:=Rn«Rs	{N,Z,C}	1
LSL {S}	Rd,Rn,#n	Logical shift left	Rd:=Rn«n	{N,Z,C}	1
LSR {S}	Rd,Rn,Rs	Logical shift right	Rd:=R»Rs	{N,Z,C}	1
LSR {S}	Rd,Rn,#n	Logical shift right	Rd:=Rn»n	{N,Z,C}	1
ASR {S}	Rd,Rn,Rs	Arithmetic shift right	Rd:=R»Rs	{N,Z,C}	1
ASR {S}	Rd,Rn,#n	Arithmetic shift right	Rd:=Rn»n	{N,Z,C}	1
ROR {S}	Rd,Rn,Rs	Rotate right	Rd:=R»Rs	{N,Z,C}	1
ROR {S}	Rd,Rn,#n	Rotate right	Rd:=Rn»n	{N,Z,C}	1
LDR	Rd,=imm	Load Immedate	Rd:=imm		2
LDR	Rt,[Rn]	Load Memory Indirect	Rt:=mem32[Rn]		2
LDR	Rt,[Rn,#offset]	Load Memory Ind. Offset	Rt:=mem32[Rn+offset]		2
LDRB	Rt,[Rn]	Load Byte unsigned	Rt:=mem8[Rn]		2
LDRB	Rt,[Rn,#offset]	Load Byte Ind. Offset	Rt:=mem8[Rn+offset]		2
LDRH	Rt,[Rn]	Load Halfword Indirect	Rt:=mem16[Rn]		2
LDRH	Rt,[Rn,#offset]	Load Halfword Ind. Offset	Rt:=mem16[Rn+offset]		2
STR	Rt,[Rn]	Store Memory Indirect	mem32[Rn]:=Rt		2
STR	Rt,[Rn,#offset]	Store Memory Ind. Offset	mem32[Rn+offset]:=Rt		2
STRB	Rt,[Rn]	Store Byte	mem8[Rn]:=Rt		2
STRB	Rt,[Rn,#offset]	Store Byte Ind. Offset	mem8[Rn+offset]:=Rt		2
STRH	Rt,[Rn]	Store Halfword Indirect	mem16[Rn]:=Rt		2
STRH	Rt,[Rn,#offset]	Store Halfword Ind. Offset	mem16[Rn+offset]:=Rt		2
PUSH	{reglist}	Push reglist on Stack			1+N
POP	{reglist}	Pop reglist from Stack			1+N
B	label	Branch undconditional	PC:=label		1+P?
B <cc>	label	Branch conditional	PC:=label		1+P?
BL	label	Branch with link	LR←PC+x+1; PC:=label		1+P?
BX	Rm	Branch indirekt	PC:=Rm-1		1+P?

Hinweise:

{S} falls ein S an den Befehl angehängt wird werden die Flags gesetzt.
op2 kann 8Bit Konstante #0xXY (auch beliebig geschoben z.B. 0xXY0) oder ein Register sein.
Rt:=mem8[adr] der Inhalt des Speichers an Adresse adr wird in die untern 8Bit des Registers Rt kopiert, die anderen Bits werden auf 0 gesetzt (bei unsigned Load).
mem8[adr]:=Rt die unteren 8Bit von Rt werden in den Speicher an Adresse adr geschrieben.

Bedingungen bei b <cc> label

<cc>	PAP	Flags	Bedeutung
EQ	=0?	Z = 1	**Equal** (to Zero)
NE	!=0?	Z = 0	**Not Equal**
CS or HS	>=	C = 1	**C**arry **S**et, **H**igher or **S**ame, unsigned
CC or LO	<	C = 0	**Lo**wer, unsigned <
MI	<0?	N = 1	**Mi**nus, Negative
PL	>=0?	N = 0	Positive or zero
VS		V = 1	**V S**et, Overflow
VC		V = 0	No overflow
HI	>	C = 1 and Z = 0	**Hi**gher, unsigned >
LS	<=	C = 0 or Z = 1	**L**ower or **S**ame, unsigned <=
GE	>=	N = V	**G**reater than or **E**qual, signed >=
LT	<	N != V	**L**ess **t**han, signed <
GT	>	Z = 0 and N = V	**G**reater **t**han, signed >
LE	<=	Z = 1 and N != V	**L**ess than or **E**qual, signed <=

Figure 15. LSL#3

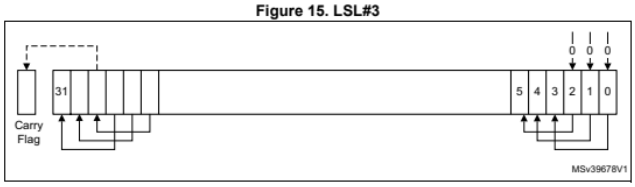


Figure 14. LSR#3

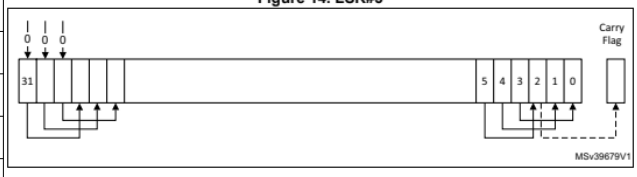


Figure 13. ASR#3

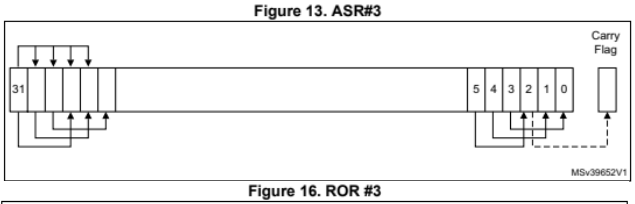
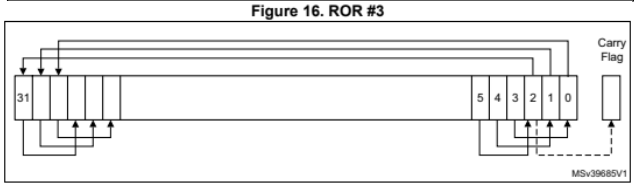
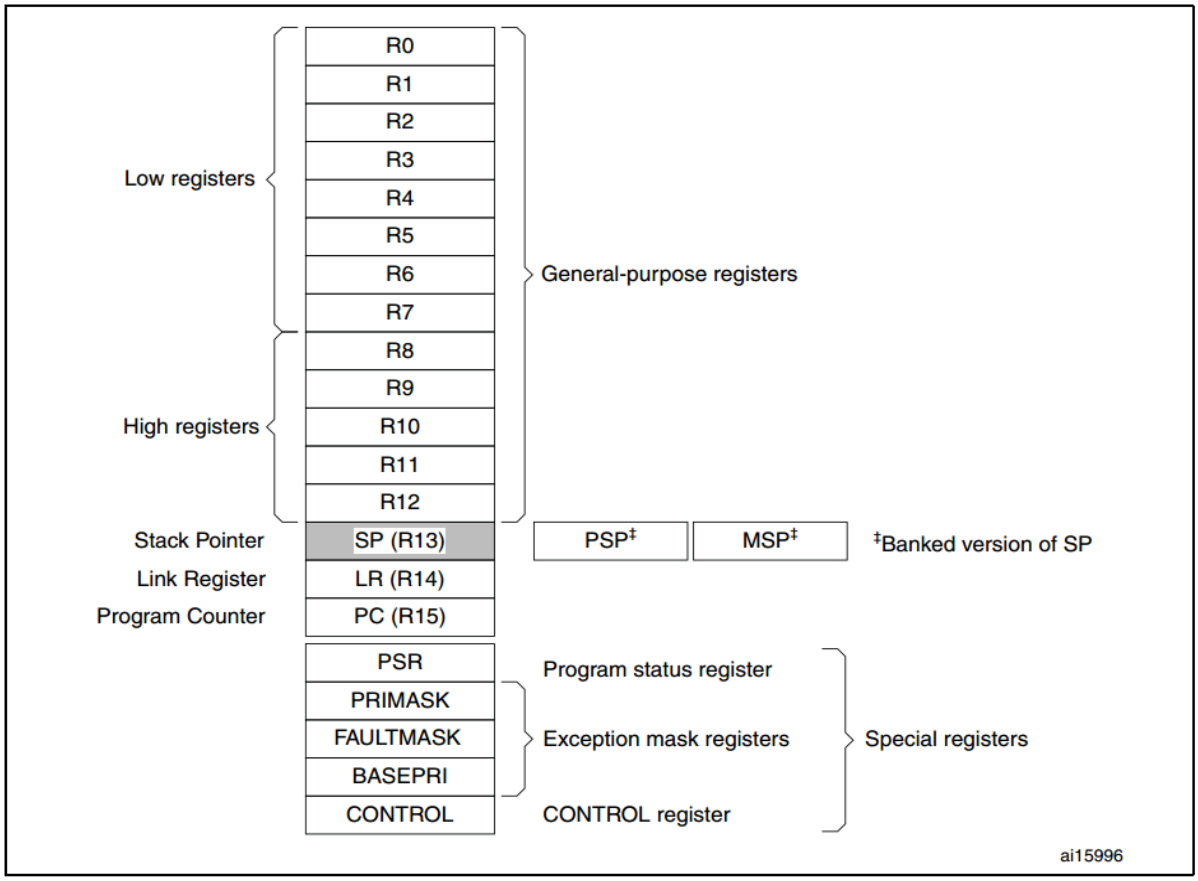
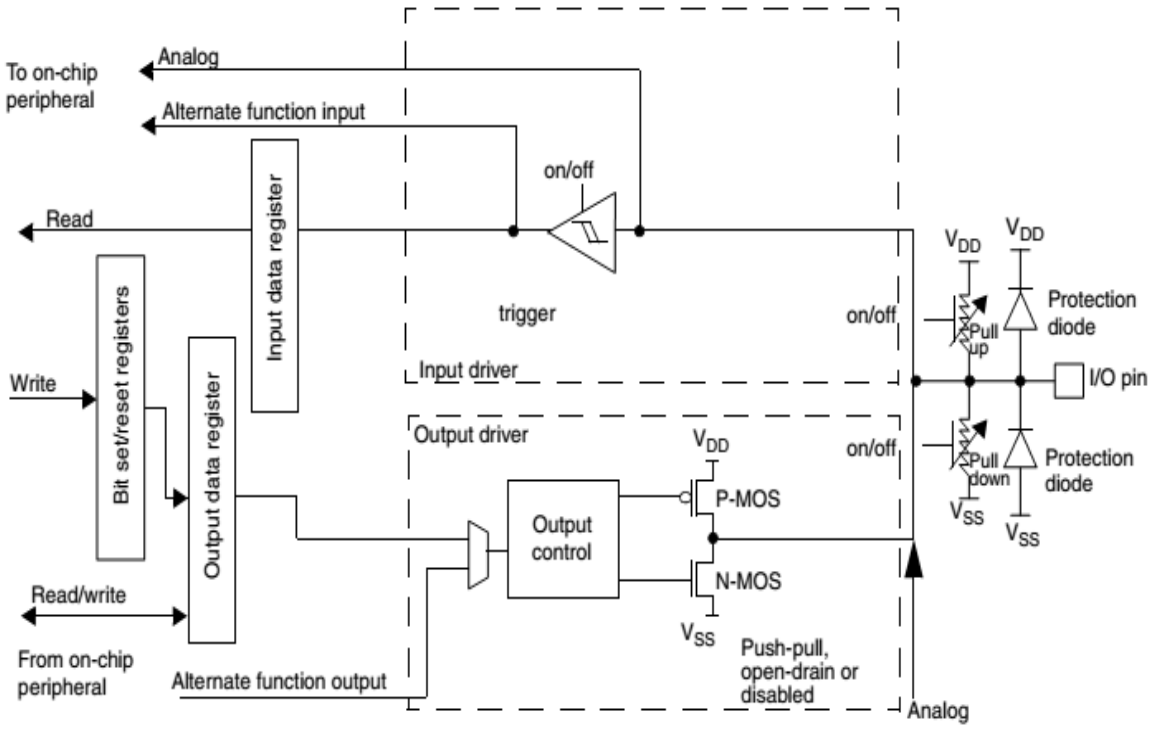


Figure 16. ROR #3







Jedem GPIO-Port ist eine Basisadresse zugeordnet und es gibt Register die seine Funktion bestimmen, Daten an den Pins lesen und ausgeben.

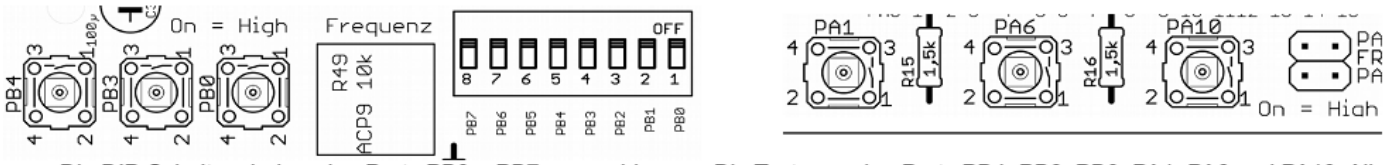
```
.equ GPIOA,0x40020000   // Basisadresse Port GPIOA
.equ GPIOB,0x40020400   // Basisadresse Port GPIOB
.equ GPIOC,0x40020800   // Basisadresse Port GPIOC
.equ MODER,0x00         // Mode des Ports
.equ OTYPER,0x04        // ?
.equ PUPDR,0x0C         // Push Pull, Open Drain or Disabled?
.equ IDR,0x10           // Input Data Register
.equ ODR,0x14           // Output Data Register
.equ BSR,0x18           // Bit Set Register
.equ BRR,0x1A           // Bit Reset Register
.equ AFRL,0x20          // ?
.equ AFRH,0x24          // ?
```

Ausgeben auf PORTC (LEDS)

```
ldr R3,=GPIOC     // lade Basisadresse in R3
mov R4,#1         // speichere Ausgabewert in R4, LED0 soll leuchten
strb R4,[R3,ODR]  // speichere die unteren 8Bit von R4 in den Speicher an die
                  // Adresse R3+ODR, also ins ODR von GPIOC
```

Einlesen von Schalter und Taster von PORTA und PORTB

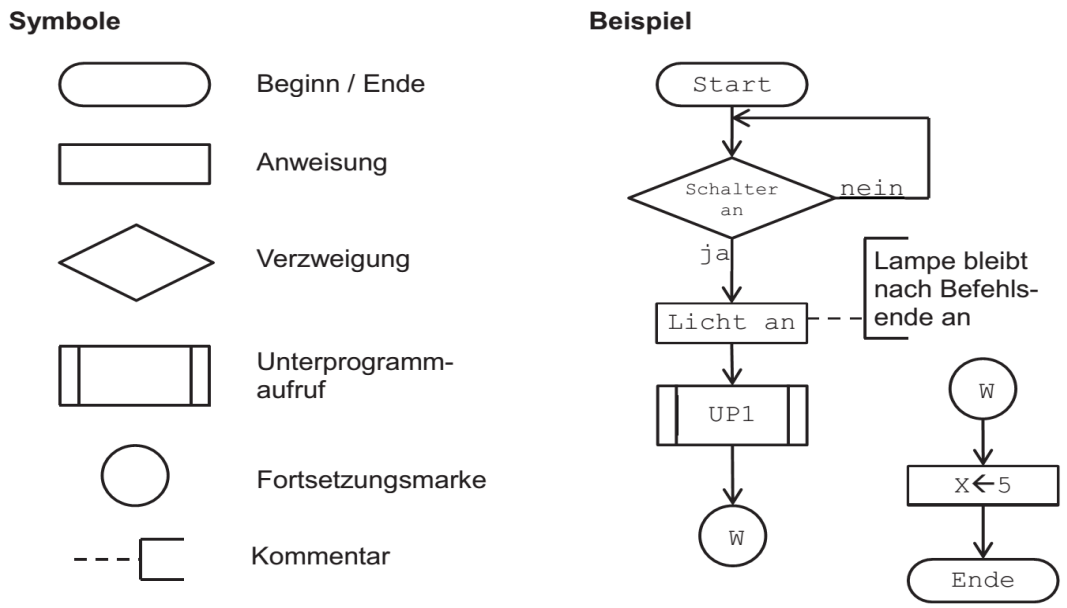
4. Taster und Schalter



Die DIP-Schalter sind an den Ports PB0 .. PB7 angeschlossen. Die Taster an den Ports PB4, PB3, PB0, PA1, PA6 und PA10. Alle Taster und Schalter sind highaktiv. In der startup.asm sind alle Taster und Schalter als Input-Pulldown vorprogrammiert.

```
ldr R3,=GPIOB     // lade Basisadresse in R3
ldr R4,[R3,IDR]   // lade aus dem Speicher von der Adresse R3+IDR in Register R4
tst R4,#0b1       // Beispiel Test ob PB0 = 1 durch bitweises Und
```

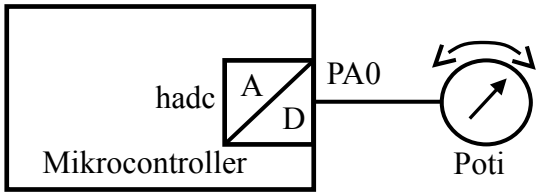
Programm Ablauf Plan PAP

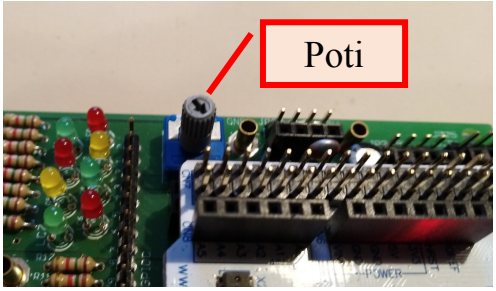


Operanden und Ausführungsprioritäten

Rang	Operator	Syntax	Richtung	Bedeutung
1	**->**	Zeiger -> Element	-->	Elementselektion
1	**[]**	Zeiger [Ausdruck], Arrayelement	-->	Indizierung
1	**()**	Ausdruck, Typ (Ausdrucksliste)	-->	Funktionsaufruf, Werterzeugung
1	**++, --**	Lvalue++, Lvalue--	-->	Post- In-,De-krement
2	**++, --**	++Lvalue, --Lvalue	<--	Prä- In-,De-krement
2	**~ , !**	~Ausdruck, !Ausdruck	<--	Komplement, Negation
2	**- , +**	-Ausdruck, +Ausdruck	<--	unäres Minus, Plus
2	**()**	(Typ) Ausdruck	<--	Typkonvertierung
2	**&**	&Variablenbezeichner	<--	unär: Adressoperator
2	*****	*Zeigerbezeichner	<--	unär: Indirektionsoperator
2	**sizeof()**	sizeof(Bezeichner)	<--	Operandengrösse in Byte
3	***, /**	Ausd. * Ausd., Ausd. / Ausd.	-->	Multiplikation, Division
3	**%**	Ausdruck % Ausdruck	-->	Modulo, Divisionsrest
4	**+, -**	Ausd. + Ausd., Ausd. - Ausd.	-->	Addition, Subtraktion
5	**<<, >>**	Ausd. << Ausd., Ausd. >> Ausd.	-->	Linksshift, Rechtsshift
6	**<, >**	Ausd. < Ausd., Ausd. > Ausd.	-->	kleiner, größer als
6	**<=, >=**	Ausd. <= Ausd., Ausd. >= Ausd.	-->	kleiner, größer gleich als
7	**==. !=**	Ausd. == Ausd., Ausd. != Ausd.	-->	gleich, ungleich
8	**&**	Ausdruck & Ausdruck	-->	bitweises Und
9	**^**	Ausdruck ^ Ausdruck	-->	bitweises EXOR
10	**|**	Ausdruck | Ausdruck	-->	bitweises Oder
11	**&&**	Ausdruck && Ausdruck	-->	logisches Und
12	**||**	Ausdruck || Ausdruck	-->	logisches Oder
13	**?:**	Ausdruck ? Ausdruck : Ausdruck	<--	Bedingte Zuweisung
14	**=**	Lvalue = Ausdruck	<--	einfache Zuweisung
14	**+=, -=, *=, /=, %=**	Lvalue op Ausdruck	<--	Kombinierte Zuweisung (arithmetisch)
14	**<<=, >>=, &=, ^=, |=**	Lvalue op Ausdruck	<--	Kombinierte Zuweisung (bitweise)
15	**,**	Ausdruck , Ausdruck	-->	Folge von Ausdrücken

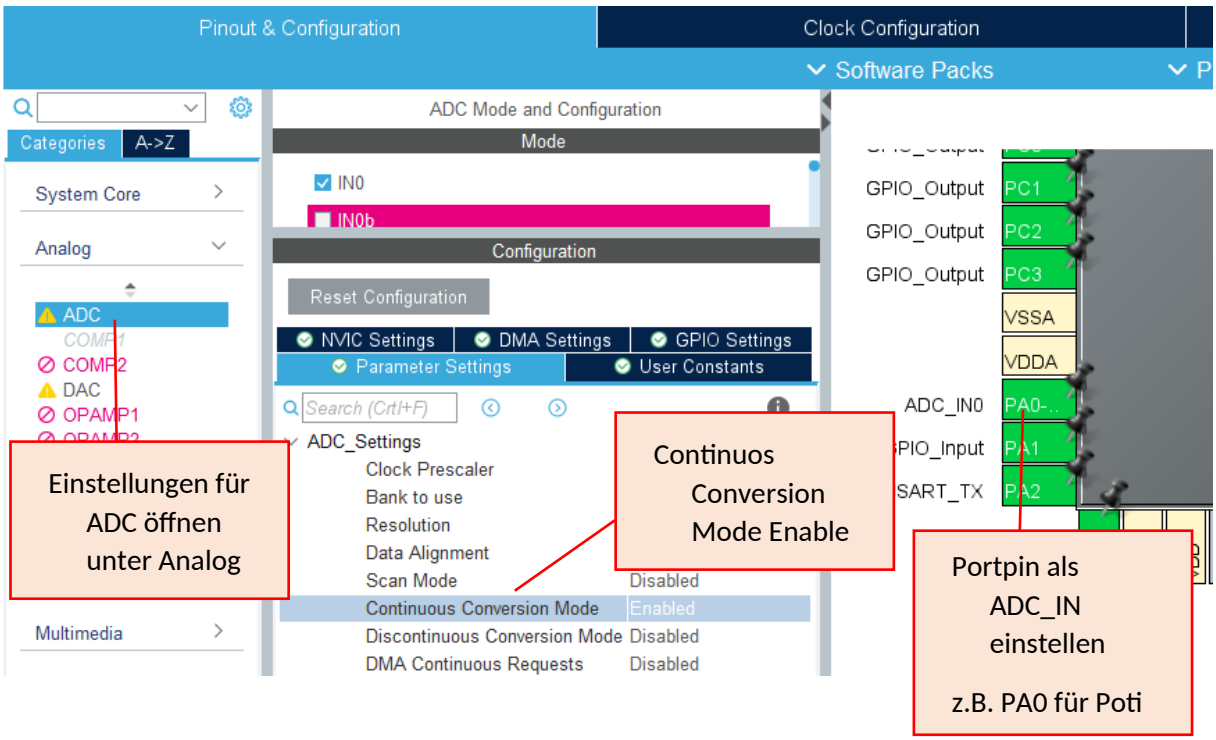
Analoge Eingabe Poti mit AD-Wandler





Analoge Sensoren wie der Einstellknopf (Poti) liefern je nach Einstellung viele verschedene Werte an den Mikrocontroller. Z.B. die Zahlen von 0 bis 4095

Voreinstellung in der Konfiguration:



Initialisierung

```
main:

  ldr R0,=hadc  //AD-Wandler starten
  bl  HAL_ADC_Start
schleife:  //Endlosschleife
…
//AD-Wandler abfragen Ergebnis in R0
  ldr  r0,=hadc  // Wert vom AD-Wandler holen
  bl  HAL_ADC_GetValue
…
b  schleife
```

LCD-i2c

LCD-display mit Backlight und i2c-Anschluss. Das Display verfügt über 2 Reihen mit je 16 Zeichen. Die Ansteuerung erfolgt mit dem Soft-i2c-Anschluss Nr 0.

Anschluss mit 4 Einzeladern:

Display	Nucleo
GND	GND
VCC	+5V
SDA	PA12 GPIO Output OpenDrain Pullup Highspeed

SCL PA11 GPIO Output Highspeed

Hinweise:
verwendet i2c-Adresse 0x27 (siehe regs.asm)
Operationen:

Initialisiert Display und i2c-Bus	`bl startLCD`
Cursor positionieren Beispiel: Cursor in die 2.Spalte der 2. Reihe positionieren	`bl` **`LCD_i2c_cursorpos`** **`Parameter:`** **`Cursorposition in R0`** **`1. Reihe: 0..0x0F`** **`2. Reihe: 0x40 .. 0x4F`** **`mov R0,#0x41`** **`bl LCD_i2c_cursorpos`**
Display löschen	`bl` **`LCD_i2c_clear`**
Text ausgeben Beispiel: "Hallo Welt"	`bl` **`LCD_i2c_textaus`** **`Parameter:`** **`R0: Adresse des auszugebenden Textes`** **`ldr R0,=meinText`** **`bl LCD_i2c_textaus`** **`am Ende des Programms:`** **`meinText:`** **`.asciz "Hallo Welt"`**

Zahl dezimal ab aktueller Cursorposition ausgeben	`mov R0,#Zahl`
	`bl` `LCD_i2c_dezaus`
	`z.b.`
	`mov R0,#1234`
	`bl LCD_i2c_dezaus`

Zahl hexadezimal ab aktueller Cursorposition ausgeben	`mov R0,#Zahl`
	`bl` `LCD_i2c_hezaus`
	`z.b.`
	`mov R0,#0x1234`
	`bl LCD_i2c_dezaus`

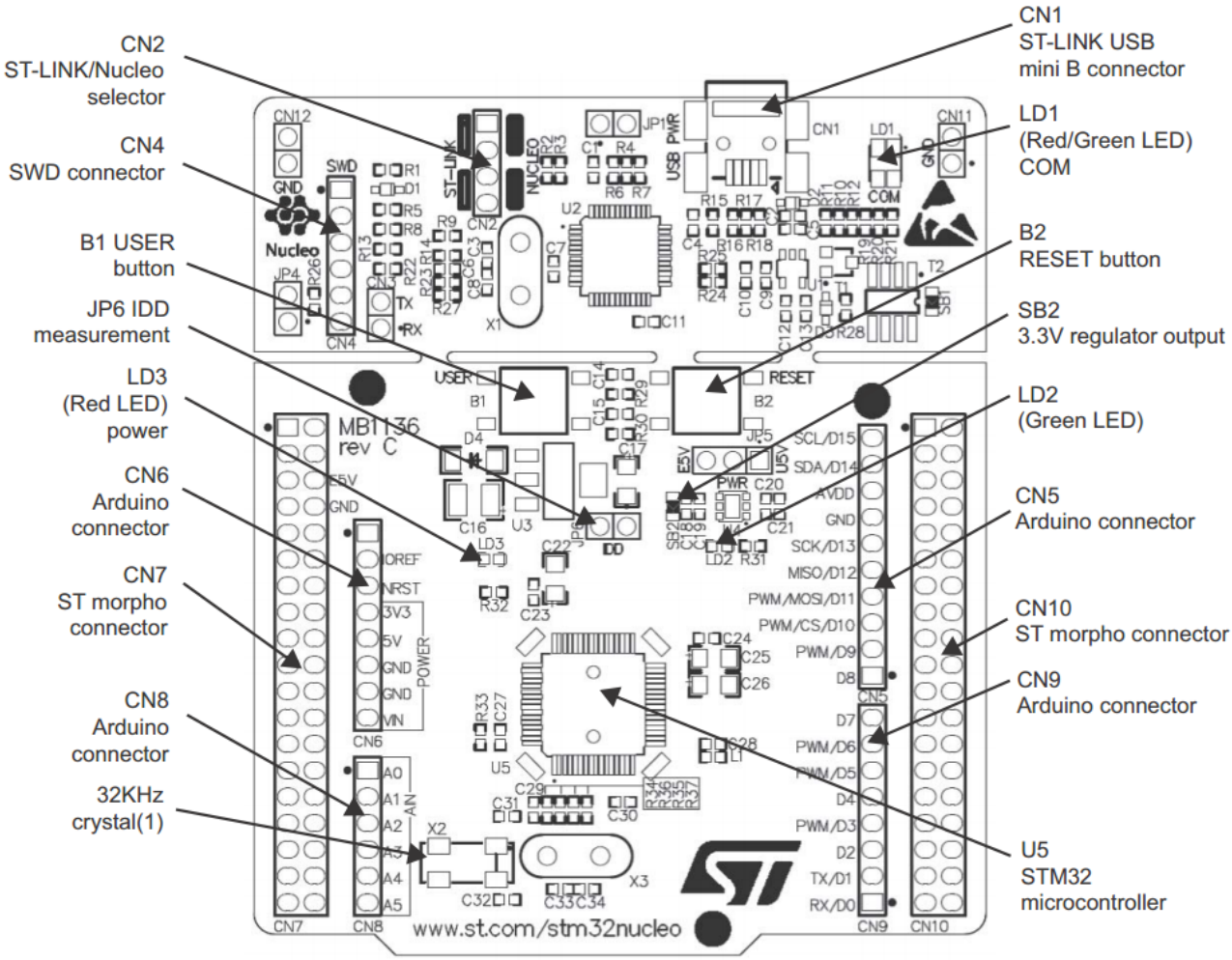
Zahl 8Bit-binär ab aktueller Cursorposition ausgeben	`mov R0,#Zahl`
	`bl LCD_i2c_binaus`
	`z.b.`
	`mov R0,#0b10101010`
	`bl LCD_i2c_binaus`

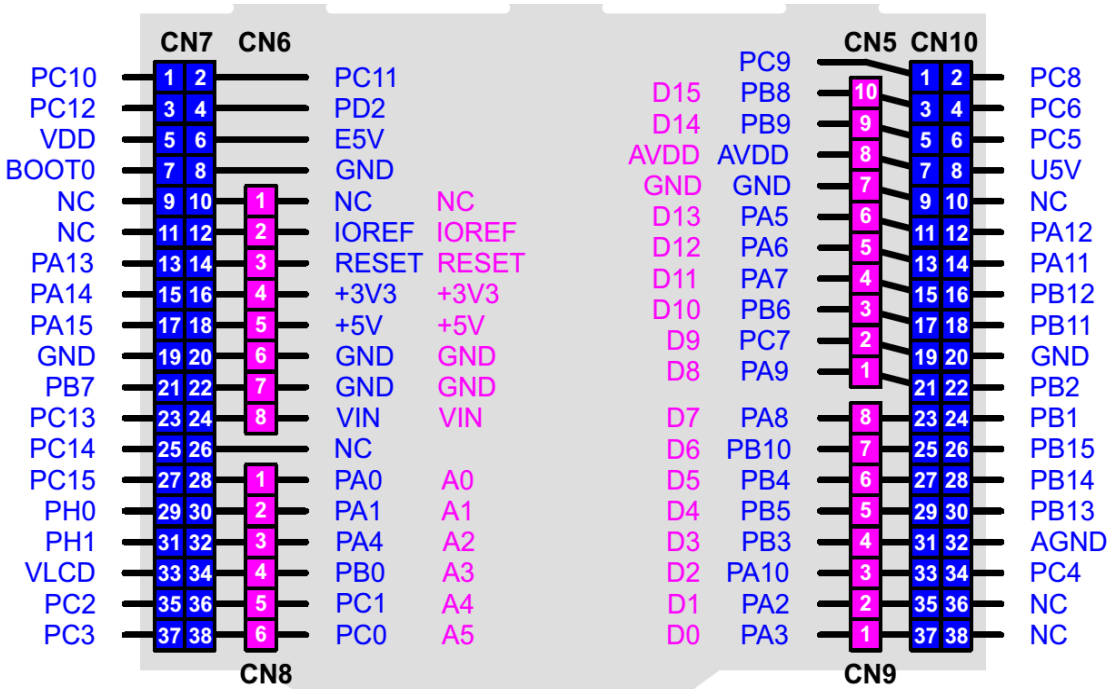
Zahl 16Bit-binär ab aktueller Cursorposition ausgeben	`mov R0,#Zahl`
	`bl LCD_i2c_bin16aus`
	`z.b.`
	`mov R0,#0xAA55`
	`bl LCD_i2c_bin16aus`

Text auf Zeile 1 ausgeben
 	
Beispiel:
"Hallo Welt"
	`bl LCD_i2c_textzeile1`
	`Parameter:`
	`R0: Adresse des auszugebenden Textes`
	
	`ldr R0,=meinText`
	`bl LCD_i2c_textzeile1`
	`am Ende des Programms:`
	`meinText:`
	`.asciz "Hallo Welt"`

Text auf Zeile 2 ausgeben	`bl LCD_i2c_textzeile2` `Parameter:`

Beispiel: "Hallo Welt"	R0: Adresse des auszugebenden Textes ldr R0,=meinText bl LCD_i2c_textzeile2 am Ende des Programms: meinText: .asciz "Hallo Welt"


CN2
ST-LINK/Nucleo
selector
CN4
SWD connector
B1 USER
button
JP6 IDD
measurement
LD3
(Red LED)
power
CN6
Arduino
connector
CN7
ST morpho
connector
CN8
Arduino
connector
32KHz
crystal(1)
CN1
ST-LINK USB
mini B connector
LD1
(Red/Green LED)
COM
B2
RESET button
SB2
3.3V regulator output
LD2
(Green LED)
CN5
Arduino connector
CN10
ST morpho connector
CN9
Arduino connector
U5
STM32
microcontroller
MB1136
rev C
www.st.com/stm32nucleo


CN7 CN6
PC10 1 2 PC11
PC12 3 4 PD2
VDD 5 6 E5V
BOOT0 7 8 GND
NC 9 10 1 NC NC
NC 11 12 2 IOREF IOREF
PA13 13 14 3 RESET RESET
PA14 15 16 4 +3V3 +3V3
PA15 17 18 5 +5V +5V
GND 19 20 6 GND GND
PB7 21 22 7 GND GND
PC13 23 24 8 VIN VIN
PC14 25 26 NC
PC15 27 28 1 PA0 A0
PH0 29 30 2 PA1 A1
PH1 31 32 3 PA4 A2
VLCD 33 34 4 PB0 A3
PC2 35 36 5 PC1 A4
PC3 37 38 6 PC0 A5
CN8
CN5 CN10
PC9
D15 PB8 10
D14 PB9 9
AVDD AVDD 8
GND GND 7
D13 PA5 6
D12 PA6 5
D11 PA7 4
D10 PB6 3
D9 PC7 2
D8 PA9 1
D7 PA8 8
D6 PB10 7
D5 PB4 6
D4 PB5 5
D3 PB3 4
D2 PA10 3
D1 PA2 2
D0 PA3 1
1 2 PC8
3 4 PC6
5 6 PC5
7 8 U5V
9 10 NC
11 12 PA12
13 14 PA11
15 16 PB12
17 18 PB11
19 20 GND
21 22 PB2
23 24 PB1
25 26 PB15
27 28 PB14
29 30 PB13
31 32 AGND
33 34 PC4
35 36 NC
37 38 NC
CN9