

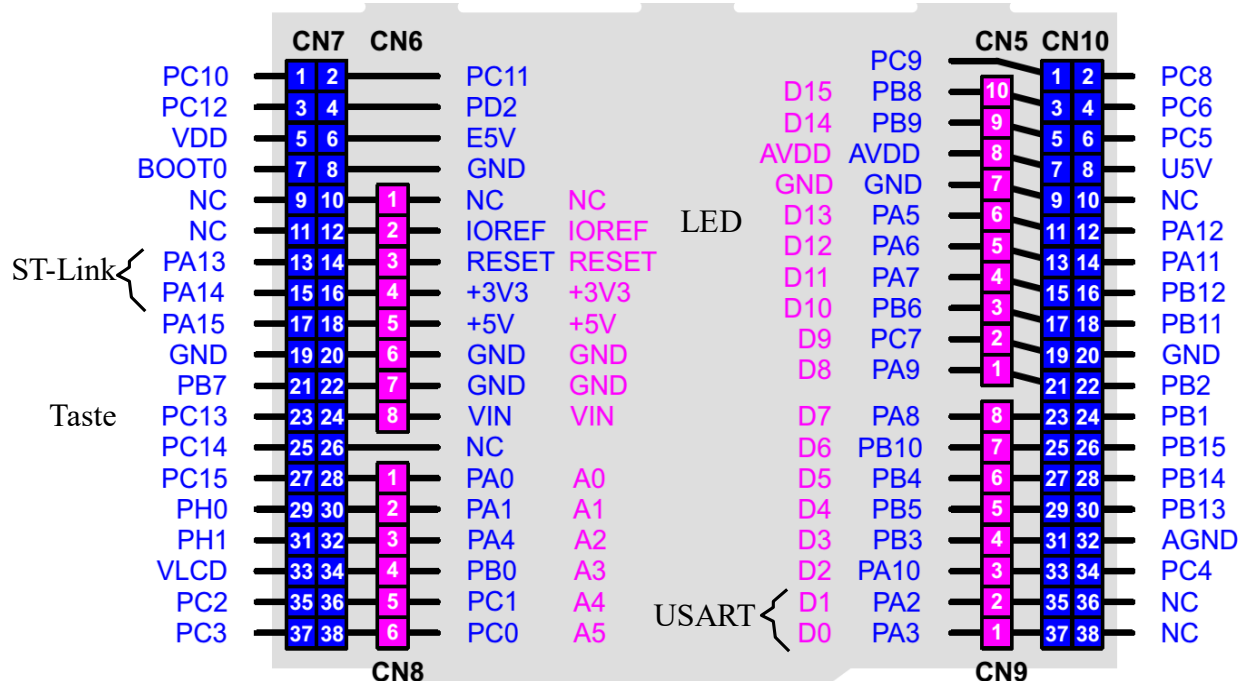
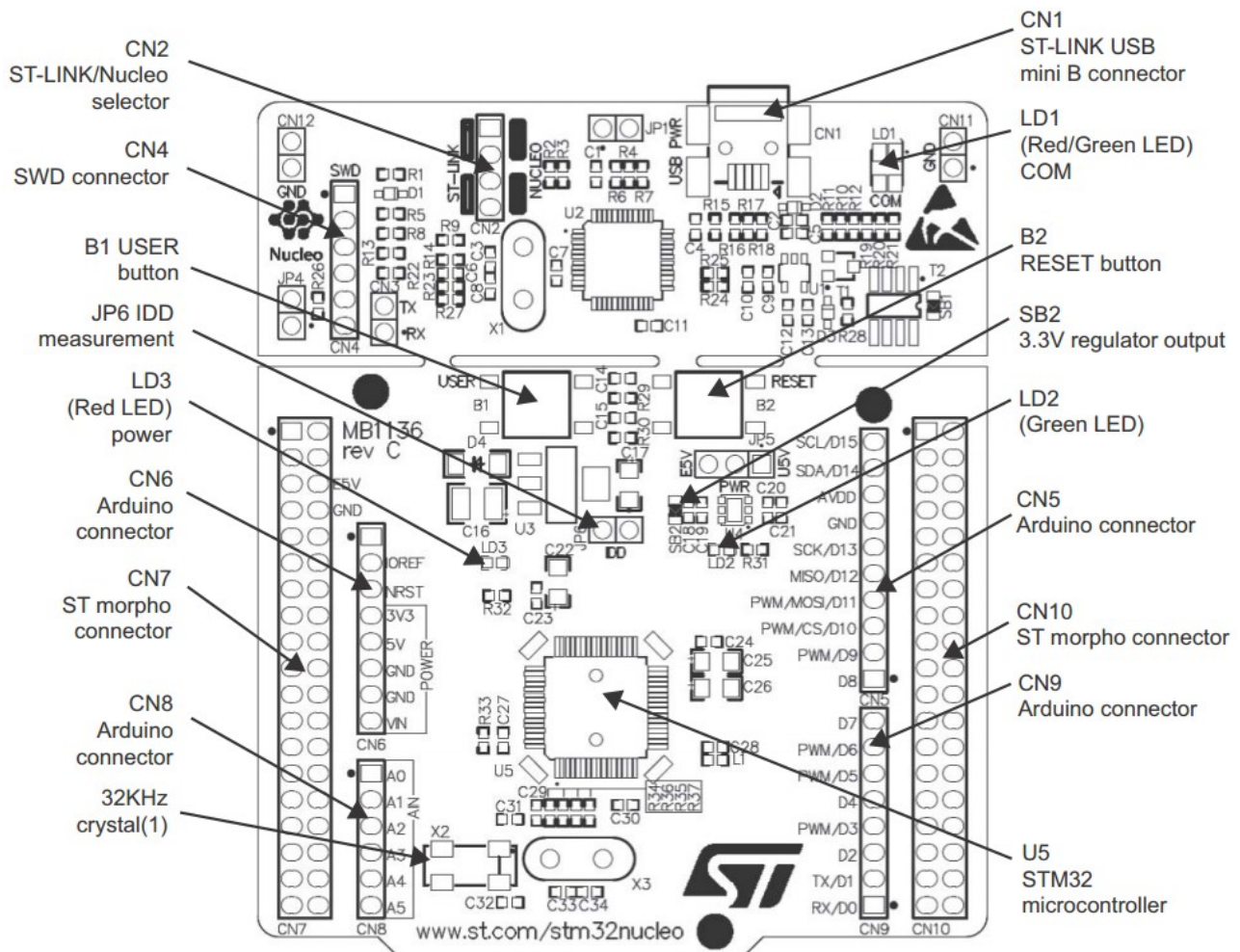
---

Inhaltsverzeichnis:

## Inhaltsverzeichnis

1. Trainingsplatine: STM32 Nucleo-64 Board.....	2
2. Programmers model .....	3
3. Instructionset.....	7
4. Wichtigste Unterprogramme in startup.asm bzw regs.s.....	9
5. Ein- und Ausgabe .....	10
5.1. Analoge Eingabe .....	11
6. LCD-i2c .....	12
7. Multifunctionshield.....	14
8. Entwicklungsumgebung Eclipse SW4STM32:.....	15

# 1. Trainingsplatine: STM32 Nucleo-64 Board



## 2. Programmers model

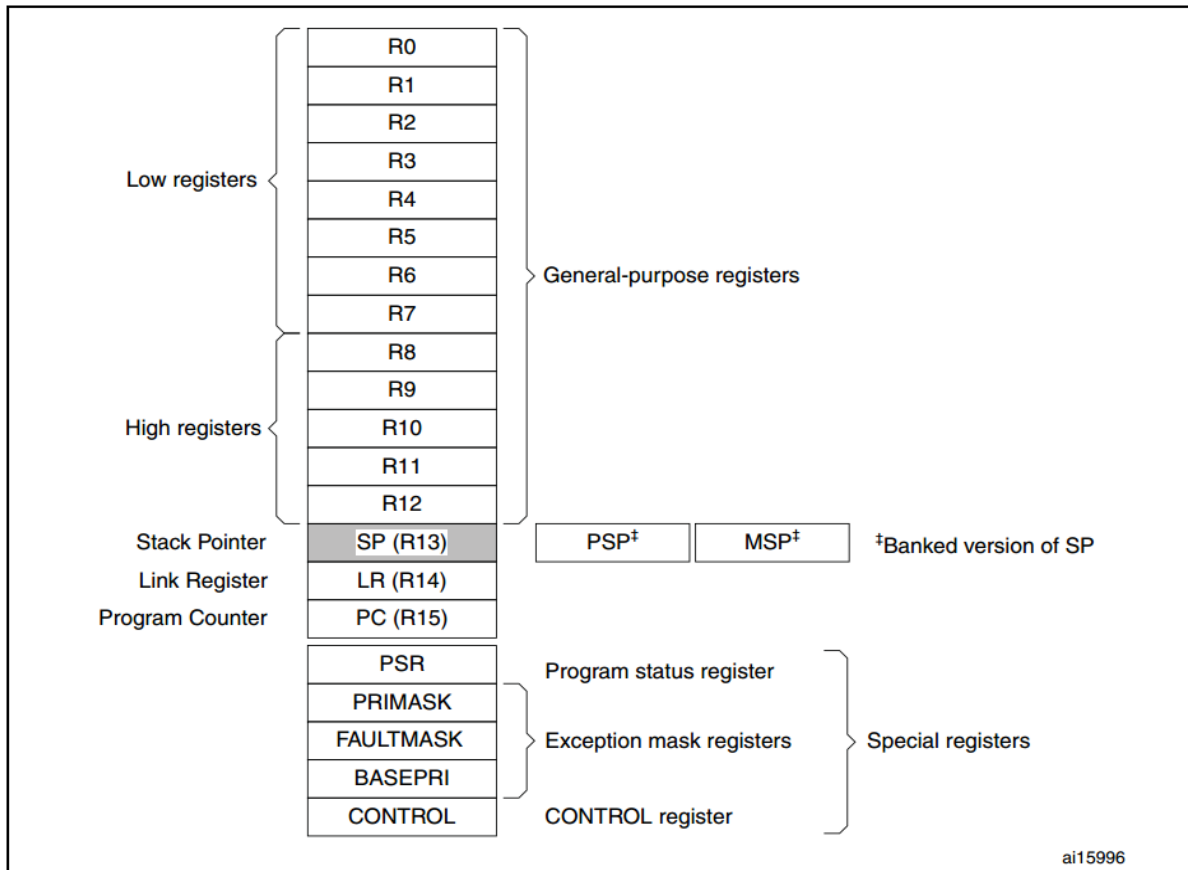


Abbildung 1: Core Registers

### General-purpose registers

R0-R12 are 32-bit general-purpose registers for data operations.

### Stack pointer

The *Stack Pointer* (SP) is register R13. In Thread mode, bit[1] of the CONTROL register indicates the stack pointer to use:

- 0 = *Main Stack Pointer* (MSP). This is the reset value.
- 1 = *Process Stack Pointer* (PSP).

On reset, the processor loads the MSP with the value from address 0x00000000.

### Link register

The *Link Register* (LR) is register R14. It stores the return information for subroutines, function calls, and exceptions. On reset, the processor loads the LR value 0xFFFFFFFF.

### Program counter

The *Program Counter* (PC) is register R15. It contains the current program address. Bit[0] is always 0 because instruction fetches must be halfword aligned. On reset, the processor loads the PC with the value of the reset vector, which is at address 0x00000004.

## ARM Cortex M3

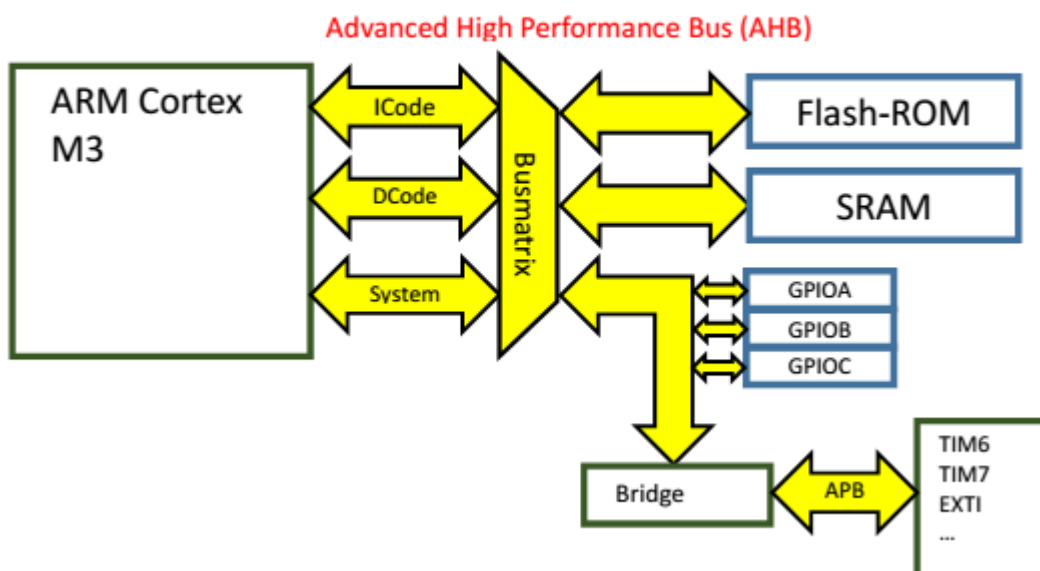
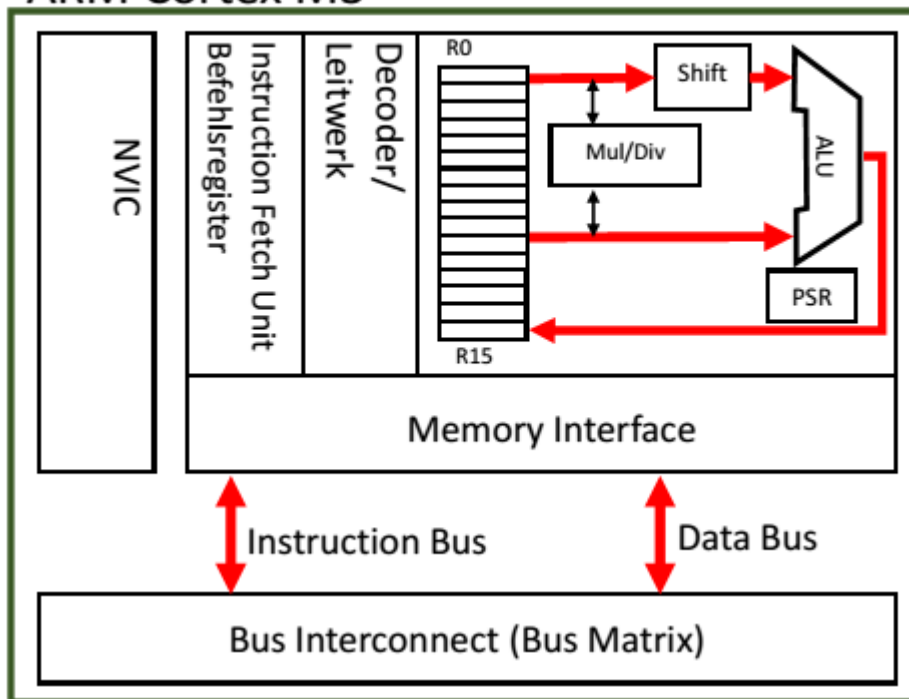


Abbildung 2: Blockdiagramm STM32L152RET6

Bits	Description
Bit 31	<b>N:</b> Negative or less than flag: 0: Operation result was positive, zero, greater than, or equal 1: Operation result was negative or less than.
Bit 30	<b>Z:</b> Zero flag: 0: Operation result was not zero 1: Operation result was zero.
Bit 29	<b>C:</b> Carry or borrow flag: 0: Add operation did not result in a carry bit or subtract operation resulted in a borrow bit 1: Add operation resulted in a carry bit or subtract operation did not result in a borrow bit.
Bit 28	<b>V:</b> Overflow flag: 0: Operation did not result in an overflow 1: Operation resulted in an overflow.
Bit 27	<b>Q:</b> Sticky saturation flag: 0: Indicates that saturation has not occurred since reset or since the bit was last cleared to zero 1: Indicates when an SSAT or USAT instruction results in saturation. This bit is cleared to zero by software using an MRS instruction.
Bits 26:0	Reserved.

*Abbildung 3: PSR Bit-Assignments*

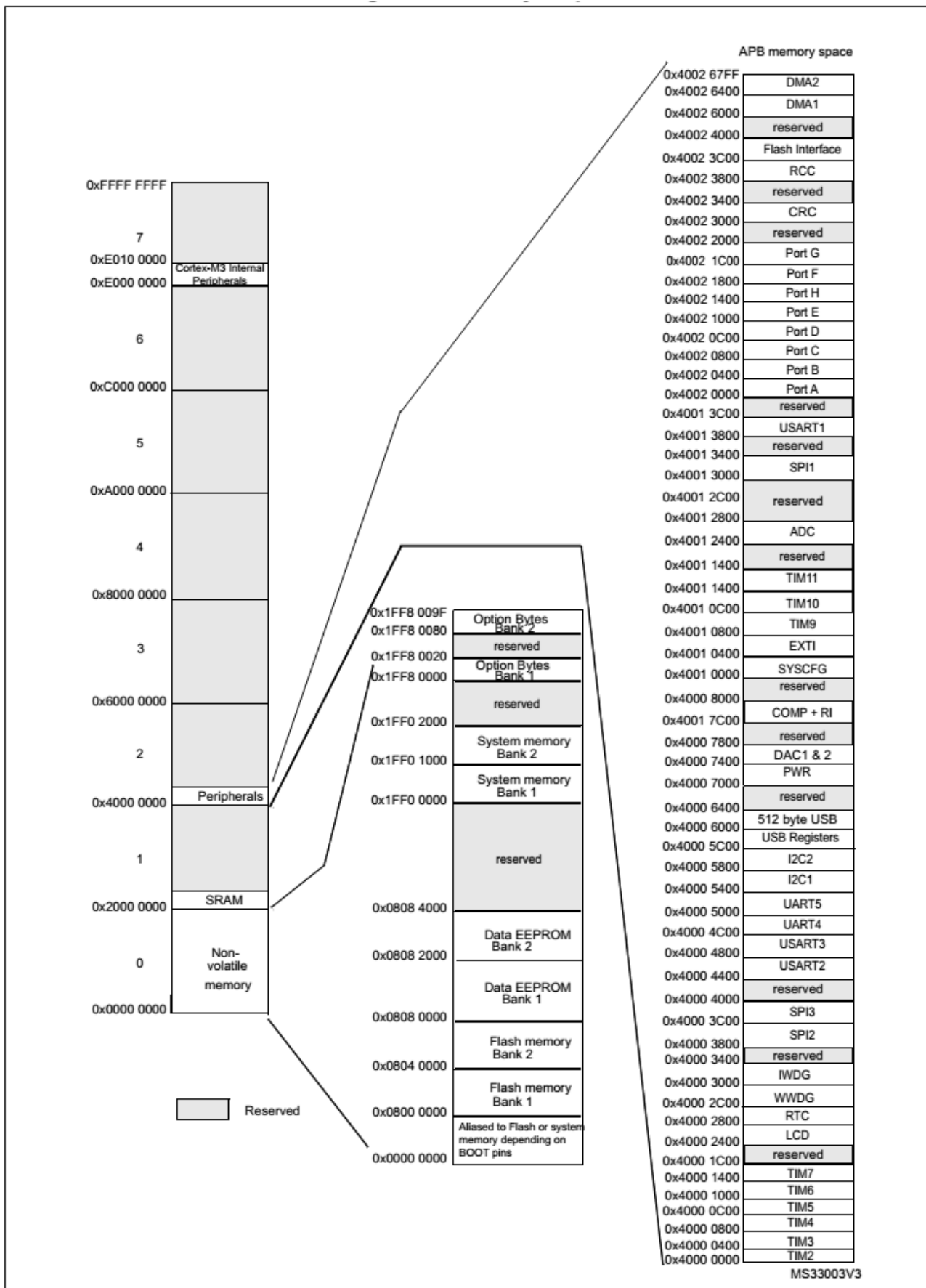


Abbildung 5: Memorymap

### 3. Instructionset

Operation	Description	Assembler	Cycles
Move	Register: Rd:=Rn	MOV Rd, Rn	1
	16-bit immediate: Rd:=imm	MOV Rd, #<imm>	1
	Immediate into top	MOVT Rd, #<imm>	1
	To PC	MOV PC, Rn	1 + P
	Move not: Rd := ! op2	MVN Rd, <op2>	1
Add	Add: Rd:=Rn+op2	ADD Rd, Rn, <op2>	1
	Add with carry: Rd:=Rn+op2+Cy	ADC Rd, Rn, <op2>	1
Subtract	Subtract: Rd:=Rn-op2	SUB Rd, Rn, <op2>	1
	Subtract with borrow	SBC Rd, Rn, <op2>	1
Multiply	Multiply: Rd:=Rn*Rm	MUL Rd, Rn, Rm	1
Divide	Unsigned: Rd:=Rn/Rm	UDIV Rd, Rn, Rm	2 to 12 <sup>[b]</sup>
Compare	Compare	CMP Rn, <op2>	1
Logical	AND: Rd:=Rn&op2	AND Rd, Rn, <op2>	1
	Test: Rn&op2 == 0	TST Rn, <op2>	1
	Exclusive OR: Rd:=Rn^p2	EOR Rd, Rn, <op2>	1
	OR: Rd:=Rn op2	ORR Rd, Rn, <op2>	1
	Bit clear	BIC Rd, Rn, <op2>	1
Shift	Logical shift left: Rd:=Rn<<imm	LSL Rd, Rn, #<imm>	1
	Logical shift left: Rd:=Rn<<Rs	LSL Rd, Rn, Rs	1
	Logical shift right: Rd:=Rn>>imm	LSR Rd, Rn, #<imm>	1
	Logical shift right: Rd:=Rn>>Rs	LSR Rd, Rn, Rs	1
	Arithmetic shift right	ASR Rd, Rn, #<imm>	1
	Arithmetic shift right	ASR Rd, Rn, Rs	1
Rotate	Rotate right	ROR Rd, Rn, #<imm>	1
	Rotate right	ROR Rd, Rn, Rs	1
Load	Word	LDR Rd, [Rn, <op2>]	2 <sup>[c]</sup>
	Word immediate	LDR Rd, =<imm>	2
	Halfword	LDRH Rd, [Rn, <op2>]	2 <sup>[c]</sup>
	Byte	LDRB Rd, [Rn, <op2>]	2 <sup>[c]</sup>
Store	Word	STR Rd, [Rn, <op2>]	2 <sup>[c]</sup>
	Halfword	STRH Rd, [Rn, <op2>]	2 <sup>[c]</sup>
	Byte	STRB Rd, [Rn, <op2>]	2 <sup>[c]</sup>
Push	Push	PUSH {<reglist>}	1 + N
Pop	Pop	POP {<reglist>}	1 + N
Branch	Bedingt <cc> siehe Tabelle	B<cc> <label>	1 or 1 + P <sup>[d]</sup>
	Unconditional	B <label>	1 + P
	With link	BL <label>	1 + P

**Hinweis:** Rd, Rn, Rs und Rm sind Register R0 .. R15

**Hinweis:** <op2> ist eine 8Bit Konstante #0xXY (auch beliebig geschoben z.B. 0xXY0) oder ein Register

**Hinweis:** Der Zusatz S an einem Add, Subtract, Logical, Shift, Rotate, Multiply oder mov-Befehl bewirkt dass die Conditioncode-Flags, entsprechend dem Ergebnis der Operation, gesetzt werden.

**Beispiel: (Zeitschleife)**

**ldr R0,=32000000/5** //Da die Pipeline unterbrochen wird benötigen die beiden Befehle  
//subs und bne zusammen 5 MZ zu je 1/32000000 s

**mov R1,#1**

**warte1s:**

**subs R0,R1** //s-Zusatz bewirkt, dass das Zero-Flag gesetzt wird.

**bne warte1s**

Tabelle der Conditioncodes: b<cc> <label>

<cc>	Flags	Meaning
EQ	Z = 1	Equal
NE	Z = 0	Not equal
CS or HS	C = 1	Higher or same, unsigned $\geq$
CC or LO	C = 0	Lower, unsigned $<$
MI	N = 1	Negative
PL	N = 0	Positive or zero
VS	V = 1	Overflow
VC	V = 0	No overflow
HI	C = 1 and Z = 0	Higher, unsigned $>$
LS	C = 0 or Z = 1	Lower or same, unsigned $\leq$
GE	N = V	Greater than or equal, signed $\geq$
LT	N $\neq$ V	Less than, signed $<$
GT	Z = 0 and N = V	Greater than, signed $>$
LE	Z = 1 and N $\neq$ V	Less than or equal, signed $\leq$

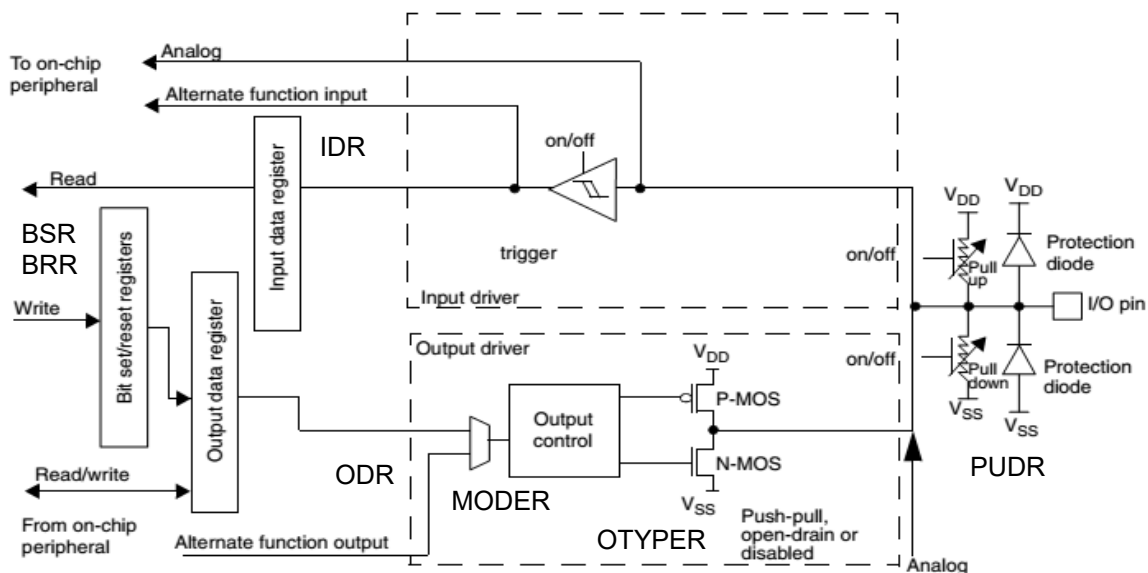


## 4. Wichtigste Unterprogramme in startup.asm bzw regs.s

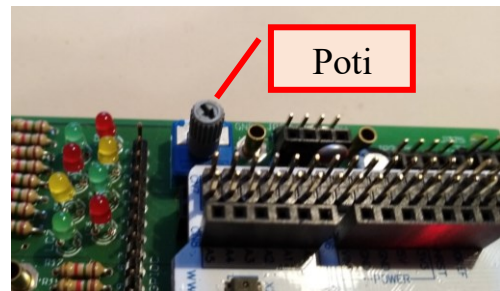
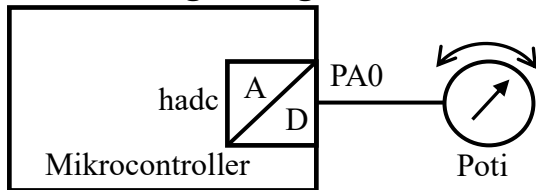
<b>startup</b>	Muss bei jedem Programm zuerst mit aufgerufen werden. Dieses Unterprogramm führt die Grundeinstellung des µController durch.
<b>Ain0</b>	Rückgabewert: r0 AD-Wandelwert (nur SW4STM32)
<b>Ain1</b>	Rückgabewert: r0 AD-Wandelwert (nur SW4STM32)
<b>Ain2</b>	Rückgabewert: r0 AD-Wandelwert (nur SW4STM32)
<b>Ain3</b>	Rückgabewert: r0 AD-Wandelwert (nur SW4STM32)
<b>Ain4</b>	Rückgabewert: r0 AD-Wandelwert (nur SW4STM32)
<b>Ain5</b>	Rückgabewert: r0 AD-Wandelwert (nur SW4STM32)
<b>wait_ms</b>	Wartezeit in ms in R0

## 5. Ein- und Ausgabe

Ausgänge wortweise (16Bit) ausgeben		
<b>Direkter Portzugriff:</b> <b>Register1:</b> R0 .. R11 <b>Register2:</b> R0 .. R11 <b>Port:</b> GPIOA, GPIOB oder GPIOC  <b>ODR:</b> <b>Output Data Register</b>	<pre>ldr Register1,=Port ldr Register2,=Wert strh Register2,[Register1,ODR] z.B. ldr R0,=GPIOA    //Port wählen ldr R1,=0x5555   //Wert festlegen strh R1,[R0,ODR] //ausgeben</pre>	<i>Port-&gt;ODR = Wert</i>  z.B.  GPIOA->ODR = 0x5555;
Ausgänge byteweise (8Bit) ausgeben		
<b>Direkter Portzugriff:</b> <b>Register1:</b> R0 .. R11 <b>Register2:</b> R0 .. R11 <b>Port:</b> GPIOA, GPIOB oder GPIOC  <b>ODR:</b> <b>Output Data Register</b>	<pre>ldr Register1,=Port mov Register2,#Wert strb Register2,[Register1,ODR] z.B. ldr R0,=GPIOA    //Port wählen mov R1,#0x55     //Wert festlegen strb R1,[R0,ODR] //ausgeben</pre>	
<b>Einzelne Bits setzen</b>	<pre>mov R0,Bit3 ldr R1,=GPIOC str R0[R1,BSR]</pre>	
<b>Einzelne Bits rücksetzen</b>	<pre>mov R0,Bit3 ldr R1,=GPIOC str R0[R1,BRR]</pre>	
Eingänge wortweise (16Bit) einlesen		
<b>Direkter Portzugriff:</b> <b>Register1:</b> R0 .. R11 <b>Register2:</b> R0 .. R11 <b>Port:</b> GPIOA, GPIOB oder GPIOC  <b>IDR:</b> <b>Input Data Register</b>	<pre>ldr Register1,=Port ldrh Register2,[Register1,IDR] z.B. ldr R0,=GPIOA    //Port wählen ldr R1,[R0,IDR]  //16Bit einlesen</pre>	<i>Wert = Port-&gt;IDR;</i> z.B.: Wert = GPIOA->IDR;



## 5.1. Analoge Eingabe



Analoge Sensoren wie der Einstellknopf (Poti) liefern je nach Einstellung viele verschiedene Werte an den Mikrocontroller.

Z.B. die Zahlen von 0 bis 4095

### 1. Voreinstellung in der Konfiguration:

The screenshot shows the STM32CubeMX configuration tool. The 'ADC Mode and Configuration' tab is selected. The 'Mode' section shows 'IN0' selected. The 'Configuration' section shows 'Continuous Conversion Mode' enabled. Three red boxes with arrows point to specific settings:

- 1. Portpin als ADC\_IN einstellen z.B. PA0 für Poti**: Points to the 'GPIO Input' section where PA0 is selected.
- 2. Einstellungen für ADC öffnen unter Analog**: Points to the 'ADC' option in the 'Analog' category.
- 3. Continuous Conversion Mode Enable**: Points to the 'Continuous Conversion Mode' checkbox, which is checked.

### 2. Initialisierung

**main:**

```
ldr      R0,=hadc           //AD-Wandler starten starten
bl      HAL_ADC_Start
```

### 3. Endlosschleife

**schleife:** //Endlosschleife

...

//AD-Wandler abfragen Ergebnis in R0

```
ldr      r0,=hadc           //Wert vom AD-Wandler holen
bl      HAL_ADC_GetValue
```

...

```
b      schleife
```

## 6. 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:

	Assembler
Initialisiert Display und i2c-Bus	<code>bl startLCD</code>
Cursor positionieren  Beispiel: Cursor in die 2.Spalte der 2. Reihe positionieren	<code>bl LCD_i2c_cursorpos</code> Parameter: Cursorposition in R0 1. Reihe: 0..0x0F 2. Reihe: 0x40 .. 0x4F  <code>mov R0,#0x41</code> <code>bl LCD_i2c_cursorpos</code>
Display löschen	<code>bl LCD_i2c_clear</code>
Text ausgeben  Beispiel: "Hallo Welt"	<code>bl LCD_i2c_textaus</code> Parameter: R0: Adresse des auszugebenden Textes  <code>ldr R0,=meinText</code> <code>bl LCD_i2c_textaus</code> am Ende des Programms: <code>meinText:</code> <code>.asciz "Hallo Welt"</code>
Zahl dezimal ab aktueller Cursorposition ausgeben	<code>mov R0,#Zahl</code> <code>bl LCD_i2c_dezaus</code> z.b. <code>mov R0,#1234</code> <code>bl LCD_i2c_dezaus</code>
Zahl hexadezimal ab aktueller Cursorposition ausgeben	<code>mov R0,#Zahl</code> <code>bl LCD_i2c_hezaus</code> z.b. <code>mov R0,#0x1234</code> <code>bl LCD_i2c_dezaus</code>



Zahl 8Bit-binär ab aktueller Cursorposition ausgeben	<pre>mov R0,#Zahl bl LCD_i2c_binaus z.b. mov R0,#0b10101010 bl LCD_i2c_binaus</pre>
Zahl 16Bit-binär ab aktueller Cursorposition ausgeben	<pre>mov R0,#Zahl bl LCD_i2c_bin16aus z.b. mov R0,#0xAA55 bl LCD_i2c_bin16aus</pre>
Text auf Zeile 1 ausgeben  Beispiel: "Hallo Welt"	<pre>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"</pre>
Text auf Zeile 2 ausgeben  Beispiel: "Hallo Welt"	<pre>bl LCD_i2c_textzeile2 Parameter: R0: Adresse des auszugebenden Textes  ldr R0,=meinText bl LCD_i2c_textzeile2 am Ende des Programms: meinText: .asciz "Hallo Welt"</pre>

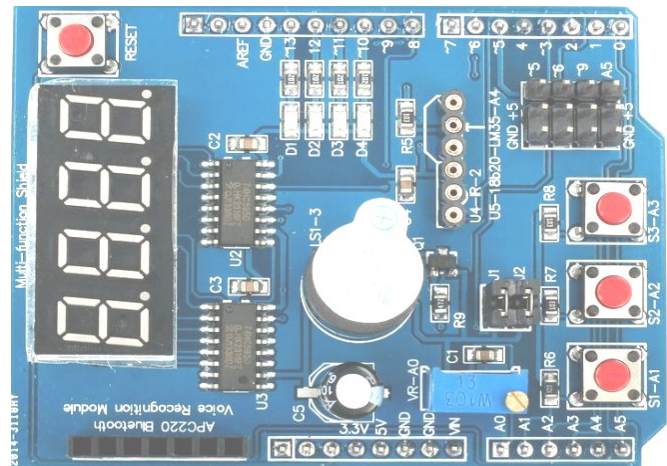
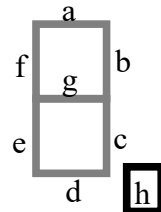
## 7. Multifunctionshield

Die Ausgabe der Daten auf das LED-Display erfolgt seriell. Die Segmente sind Low-Aktiv.

Tabelle:

```
char siebenseg[10]=
{0b00000011,0b10011111,0b00100101,
0b00001101,0b10011001,0b01001001,
0b01000001,0b00011111,0b00000001,
0b00001001};
```

segmente: 0Babcdefgh



Anschlüsse:

Seriell Data SDI:	D8 (PA9)	Konfigurieren als Digitalausgang in CubeIDE
Shift Clock SFTCLK:	D7 (PA8)	Konfigurieren als Digitalausgang in CubeIDE
Latch Clock LCHCLK:	D4 (PB5)	Konfigurieren als Digitalausgang in CubeIDE

Poti:	A0 (PA0)	Konfigurieren als ADC_IN in CubeIDE
Taste S1:	A1 (PA1)	Konfigurieren als Digitaleingang in CubeIDE
Taste S2:	A2 (PA4)	Konfigurieren als Digitaleingang in CubeIDE
Taste S3:	A3 (PB0)	Konfigurieren als Digitaleingang in CubeIDE

(alle aktiv Low)

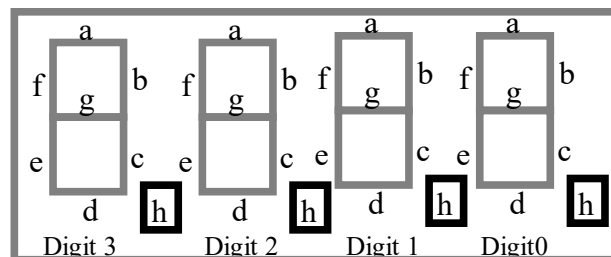
LED D1:	D13 (PA5)	Konfigurieren als Digitalausgang in CubeIDE
LED D2:	D12 (PA6)	Konfigurieren als Digitalausgang in CubeIDE
LED D3:	D11 (PA7)	Konfigurieren als Digitalausgang in CubeIDE
LED D4:	D10 (PB6)	Konfigurieren als Digitalausgang in CubeIDE
Summer:	D3 (PB3)	Konfigurieren als Digitalausgang in CubeIDE

(alle aktiv Low)

Operationen:

	Assembler
Initialisierung des Displays	bl MFS_serial_init
Daten an das Display schicken	mov r0,#Data bl MFS_sendWord
Beispiel: 2 auf Digit 0 = 0x1025	Beispiel: mov r0,#0x1025 bl MFS_sendWord

Format von Data:



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Digit 3	Digit 2	Digit 1	Digit 0	0	0	0	0	a	b	c	d	e	f	g	h

Hinweis: Das Display arbeitet im Multiplexbetrieb!

## 8. Entwicklungsumgebung Eclipse SW4STM32:

Neues Projekt anlegen:

- 1 Eclipse starten und Workspace wählen.
- 2 File -> New -> C-Project
- 3 Projektname eintragen
- 4 Ac6STM32 MCU Project
- 5 -> next
- 6 Configuration Debug und Release
- 7 -> next
- 8 Target Configuration
  - MCU:
    - Series: STM32L1
    - MCU: STM32L152RETx
  - Board:
    - Series STM32L1
    - Board NUCLEO-L152RE
- 9 -> Next
- 10 Standard Peripheral Library wählen
- 11 die 5 Dateien aus der Vorlage in den Ordner src in Eclipse kopieren
- 12 in mainasm befindet sich das Assemblerprogramm
- 13 Erstes Starten des neuen µController-Programms mit:
  - Rechtsklick auf das Projekt
  - Run As
  - AC6 Stm32 C/C++ Application

