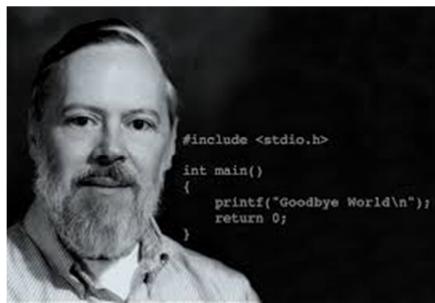


Digitale Systeme und Computersysteme



Dennis Richie (1941-2011)

Skriptum zur Lehrveranstaltung

Einführung in die C-Programmierung des ARM Cortex-M3

Dipl. Ing. Josef Reisinger

September 2015

Dieses Skriptum dient als Grundlage für den Unterricht des Gegenstands Digitale Systeme (DIC) an den 4. Klassen der HTL Hollabrunn Abteilung Elektronik. Es kann und will keine Bücher ersetzen und dient hauptsächlich zur Unterrichtsbegleitung und ist nur bedingt zum Selbststudium geeignet.

Inhaltsverzeichnis

1	Einführung C- Programmierung des ARM Cortex-M3.....	4
1.1	Einführung:.....	4
1.2	Datentypen in C.....	4
1.3	Aufbau des Headerfiles STM32F10X.h	5
1.3.1	Beispiel: Konfigurieren der Portleitung PB8 für LED0 der LED Schalterplatine	6
1.3.2	Beispiel: Setzen und Löschen einer Leitung unter Benutzung von Bit Banding:.....	6
1.4	Beispielprogram #1 – Lauflicht	7
1.5	Beispielprogram #2 – Flanke Detektion, Ausgabe Variable UART.....	10
1.5.1	Entprellen von Schaltern.....	13
1.6	Beispielprogram #3 – Serielle Ausgabe(SoftUart).....	14
1.7	Beispielprogram #4 – Ampel	15
2	ADC des STM32F10X.....	17
2.1	Interner Aufbau	17
2.1.1	Kanalauswahl:	17
2.1.2	Single Conversion Mode:.....	18
2.1.3	Continuous Conversion Mode:.....	18
2.1.4	Scan Mode:	18
2.1.5	Timing Diagramm	19
2.1.6	ADC Register.....	19
2.2	Beispielprogram #5 – Einlesen eines Analogwerts über ADC1.....	22
3	Reset and Clock Control (RCC) des ARM Cortex-M3	23
3.1	Beispielprogram – Set SystemClock.....	24
3.2	Register Clock Control.....	25
4	Interrupt Programmierung des ARM Cortex-M3	27
4.1	Einführung.....	27
4.2	Blockschaltbild	27
4.3	NVIC Konfiguration.....	27
4.4	Beispielprogram #6 – Interrupt UART.....	29
4.5	Interruptnummern.....	30
4.6	Namen der Interrupt Service Routinen	32
4.7	Externe Interrupts.....	33
4.8	Beispielprogram #7 – Externer Interrupt	34
5	Timer des ARM Cortex-M3.....	36
5.1	Übersicht.....	36
5.2	Blockschaltbild eines Timers	37
5.3	Konfiguration eines Timers	37
5.3.1	Clock Selection.....	37
5.3.2	Prescaler	38
5.3.3	CounterModi	38
5.3.4	Repetition Counter.....	40
5.4	Interrupt Konfiguration der Timer	40
5.5	Timer Zeitberechnung	41
5.5.1	Messung und Berechnung der Taktfrequenz	41
5.5.2	Zeitberechnung.....	41
5.6	Beispielprogram #8 – Lauflicht mit Timer Interrupt.....	43
5.7	Capture Compare Channel.....	45
5.7.1	Interrupt Konfiguration für Capture/ Compare Einheit	45
5.7.2	Input Capture.....	45
5.8	Beispielprogram #9 –Timer Capture Demoprogramm	47
5.8.1	Output Compare	49
5.8.2	Countermodi für Output Compare	50
5.9	Beispielprogram #10 –Output Compare mit PWM Demoprogramm.....	51
6	DMA Controller des ARM Cortex-M3	53
6.1	Allgemeiner Aufbau	53
6.2	Konfiguration eines DMA Transfers	54

6.3	Beispielprogram #11 –DMA Controller und ADC Demoprogramm.....	56
7	CMSIS-Cortex Mikrocontroller Software Interface Standard.....	60
8	StdPeriph Drivers für STM32F10X Microcontroller	62
8.1	Beispielprogram #6 – Interrupt UART.....	62
8.2	Beispielprogram #7 – Externer Interrupt.....	64
8.3	Beispielprogram #8 – Lauflicht mit Timer Interrupt.....	66
8.4	Beispielprogram #9 –Input Capture Demoprogramm.....	68
8.5	Beispielprogram #10 –Output Compare mit PWM Demoprogramm.....	70
8.6	Beispielprogram #11 –DMA Controller und ADC Demoprogramm.....	72

1 Einführung C- Programmierung des ARM Cortex-M3

1.1 Einführung:

Für die Programmierung von **embedded Systems** gibt es verschiedene Programmiersprachen. Die klassische Programmiersprache für embedded Systems ist **Assembler**. Assemblersoftware ist nach geringer Einarbeitungszeit relativ einfach und rasch zu erstellen. Der Vorteil der Sprache liegt darin, dass die entsprechend übersetzte Software auf den jeweiligen Prozessortyp ca. **2–5 mal schneller** abgearbeitet wird als entsprechende Hochsprachenprogramme gleicher Funktionalität. Ein weiterer Vorteil der Assemblersprache besteht darin, dass der **Platzbedarf im Programmspeicher** erheblich geringer ist.

Der große Nachteil von Assemblersoftware ist jedoch die mangelnde **Wiederverwendbarkeit der Software** für andere Projekte. Jahre an Entwicklungsarbeit gehen verloren, wenn für die nächste Produktversion der Prozessor von einer anderen Firma kommt. Assemblercode ist in den meisten Fällen von Hersteller zu Hersteller unterschiedlich.

Auch die **Übersichtlichkeit** der Software und die Möglichkeiten der **Strukturierung** sind bei Assembler beschränkt. So ist es z.B. ohne trickreiche Konstruktionen nicht möglich lokale Variablen anzulegen und diese selbstdokumentierend zu benennen. Ein Assemblerprogramm ist normalerweise komplizierter und damit schwerer zu **warten** als ein C- Programm mit gleicher Funktionalität. Wächst ein Assemblerprogramm so ergeben sich neue Probleme. Die **Schnittstellen** zwischen den einzelnen Programmen werden komplizierter und schwerer zu handhaben.

Aus diesen Gründen werden in kommerziellen Entwicklungen zumeist **Hochsprachen** eingesetzt, da unter Einhaltung gewisser Regeln der **Portierungsaufwand** auf andere Prozessoren mit vertretbaren Aufwand gelingen kann.

Von den höheren Programmiersprachen kommt C in Verbindung mit Mikrocontrollern am häufigsten zum Einsatz, da es der Hardware am nächsten steht. Dies zeigt sich bei Pointern, bei der Manipulation von Bits und bei Bitfeldern. Die Möglichkeiten zur Strukturierung von Daten und Code sind einerseits umfangreich genug, andererseits können diese effizient in Code umgesetzt werden.

Der ANSI Standard von 1987 bildet das Fundament für eine portable C- Entwicklung. Jeder ANSI C- Compiler sollte den Code eines anderen ANSI C- Compilers verstehen.

1.2 Datentypen in C

Da die Implementierung der Basistypen nicht im Detail von der Sprache „C“ definiert ist, sind dadurch dem Compilerhersteller Freiheiten gegeben. Der Standard legt nur fest, dass `char` <= `short` `int` <= `int` <= `long` <= `long long` und `float` <= `double` <= `long double`.

In der folgenden Tabelle ist die Implementierung für den ARM „C“Compiler der Fa. ARM (Keil) dargestellt.

Datentyp	Größe in Byte	Wertebereich	Bemerkung
Char	1	-128...+127	standardmäßig ist der typ char unsigned !
unsigned char	1	0...255	
Short	2	-32768....+32767	
Int	4	-2 ³¹ ...+2 ³¹ -1	standardmäßig ist der typ int signed !
unsigned int	4	0...+2 ⁶⁴ -1	
long	4	-2 ³¹ ...+2 ³¹ -1	
unsigned long int	4	0...+2 ⁶⁴ -1	
long long	8	-2 ⁶³ ...+2 ⁶³ -1	
Float	4	-3,4.10 ⁻³⁸ ...3,4.10 ³⁸	IEEE Format
Double	8		IEEE Format
long double	8		IEEE Format
Void			

Bei einigen Situationen ist es manchmal sinnvoll zu wissen, wie Daten im Speicher abgelegt werden. Eventuell muß ein Speicherdump gelesen oder ein Hardwarebaustein angesprochen werden. Bei den Datentypen, die größer 1 Byte sind (also int, long int, usw.) gibt es 2 Möglichkeiten die Bytes im Speicher anzutragen.

- Speicherung beginnt mit dem niedrigstwertigen Byte und endet mit höchstwertigen Byte (= **Big Endian** bzw. Motorola order)
- Speicherung beginnt mit dem höchstwertigen Byte und endet mit dem niedrigstwertigen Byte (= **Little Endian** bzw. Intel Order)

Der ARM Compiler der Fa. Keil verwendet standardmäßig für das Cortex-M3 device die **Little-Endian** Konvention.

1.3 Aufbau des Headerfiles STM32F10X.h

Für die low level Zugriffe auf die Register der Peripherieeinheiten stellt der Hersteller(Vendor) des Cortex-M3 STM32F103RB ein Headerfile bereit. Die Bezeichnung bei ST-Microelectronics lautet STM32F10X.h Der Zugriff auf die Register der einzelnen Peripherieeinheiten erfolgt erfolgt über **Zeiger (Pointer) auf Strukturen**. Im Folgenden wird dies exemplarisch für den PortB des Cortex-M3 gezeigt

```
typedef unsigned int          uint32_t;
typedef struct
{
    __IO uint32_t CRL;
    __IO uint32_t CRH;
    __IO uint32_t IDR;
    __IO uint32_t ODR;
    __IO uint32_t BSRR;
    __IO uint32_t BRR;
    __IO uint32_t LCKR;
} GPIO_TypeDef;
```

Da die Register der Ports adressmäßig im Speicher hintereinander liegen wird ein C- Struktur GPIO_TypeDef definiert.

```
#define PERIPH_BASE      ((uint32_t)0x40000000)
#define APB2PERIPH_BASE  (PERIPH_BASE + 0x10000)
#define GPIOB_BASE        (APB2PERIPH_BASE + 0x0C00)
#define GPIOB             ((GPIO_TypeDef *)GPIOB_BASE)
#define PERIPH_BB_BASE   ((uint32_t)0x42000000)
```

Im folgenden wird ein Pointer (z.B.: GPIOB) definiert, der auf die Basisadresse des jeweiligen Ports zeigt und ein Pointer auf ein GPIO_TypeDef Struktur ist. Über diesen Pointer kann auf die einzelnen Strukturelemente zugegriffen werden.

1.3.1 Beispiel: Konfigurieren der Portleitung PB8 für LED0 der LED Schalterplatine

Bevor einer Portleitung beim ARM Cortex-M3 verwendet werden kann muss der entsprechende Port mit dem Takt versorgt werden und die entsprechende Portleitung konfiguriert werden. Der folgende Programmcode konfiguriert die Portleitung PB8 in den Modus Push Pull.

```
int temp;           // temp Varibale definieren

RCC->APB2ENR |= RCC_APB2ENR_IOPBEN;    // enable clock for GPIOB
temp = GPIOB->CRH;        // aktuelle Konfiguration in temp Varibale sichern
temp = &= 0xFFFFFFFF0; // Konfigurationbits für Leitung PB8 löschen
temp = |= 0x3;   // Leitung PB8 als Push Pull Output definieren
GPIOB->CRH = temp;    // Konfiguration übernehmen
```

Beispiel: Setzen und Löschen einer Portleitung über Peripherieregister

Mit folgendem Code in der Programmiersprache C kann nun das Bit8 des PortB (=LED0) einfach gesetzt werden

```
GPIOB->ODR |= 0x0100; oder
GPIOB->BSRR = 0x0100; /* besser: atomare Bit Set Operation !!*/
```

Mit folgendem Code in der Programmiersprache C kann nun das Bit8 des PortB (=LED0) einfach gelöscht werden

```
GPIOB->ODR &= 0xFEFF; oder
GPIOB->BRR = 0x0100 /* besser: atomare Bit Reset Operation !!*/
```

1.3.2 Beispiel: Setzen und Löschen einer Leitung unter Benutzung von Bit Banding:

Das Ein- und Ausschalten der Portleitung könnten wir genauso unter Verwendung von Bit Banding des ARM Cortex-M3 erreichen. Zu diesem Zweck müssen wir zu einer gegeben Speicheradresse die entsprechende Bit Alias Adresse berechnen. Diese kann wie schon in Kapitel Bit Banding erklärt wurde mithilfe folgender Formel berechnet werden.

Adresse im Bit Band Alias Bereich = Bit Band Alias Basis Adresse + Bit Word Offset
Bit Word Offset = Byte offset von Bit Band Basis * 0x20 + Bitnummer * 4

In der Programmiersprache C kann dies elegant mit folgendem **Makro** gelöst werden

```
// Calc Bit Band Adress from peripheral address:
// a = peripheral address, b = Bit number
```

```
#define BITBAND_PERI(a,b) ((PERIPH_BB_BASE + (a-PERIPH_BASE)*32 + (b*4)) )

// Bit Band Alias Adresse Portleitung PB8
#define LED0 *((volatile unsigned long *) (BITBAND_PERI(GPIOB_ODR, 8)))
```

Mit folgendem Code in der Programmiersprache C kann nun das Bit8 des PortB (=LED0) einfach gesetzt werden:

```
LED0 = 1; // LED0 einschalten
```

Mit folgender Anweisung Programmiersprache C kann LED0 gelöscht werden

```
LED0 = 0; // LED0 ausschalten
```

Frage: Was bedeutet das Schlüsselwort „volatile“ in C ?

1.4 Beispielprogramm #1 – Lauflicht

```
/*********************************************
/* (C) Copyright HTL - HOLLABRUNN 2009-2009 All rights reserved. AUSTRIA */
/*
/* File Name: LAUFLICHT.c
/* Autor: Josef Reisinger
/* Version: V1.00
/* Date: 11/30/2009
/* Description: Lauflicht mit variabler Zeit
/*********************************************
/* History: V1.00 creation
/*********************************************
#include <stm32f10x.h>

/* _____ FUNCTION-PROTOTYPES _____ */
void init_leds_switches (void);
void set_leds (char);
char get_switches(void);
void wait_ms(int);
void wait(void);

/*********************************************
/*      U N T E R P R O G R A M M:    init_leds_switches
/*
/* Aufgabe: Initialisiert Portleitungen für LED/Schalterplatine
/* Input:
/* return:
/*********************************************
void init_leds_switches(void) {

int temp;

// enable clock for GPIOA (APB2 Peripheral clock enable register)
RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;
// enable clock for GPIOB (APB2 Peripheral clock enable register)
RCC->APB2ENR |= RCC_APB2ENR_IOPBEN;

// Schalter auf Led / Schalterplatine konfigurieren
temp = GPIOA->CRL;
temp &= 0x00000000; // PA7-PA0 Konfigurationsbits löschen
temp |= 0x88888888; // PA7-PA0 als Input definieren
GPIOA->CRL = temp;
GPIOA->BSRR = 0x00FF; // PA7-PA0 internen Pull Up aktivieren (ODR)
// LEDs auf Led / Schalterplatine
temp = GPIOB->CRH;
temp &= 0x00000000; // PB15-PB8 Konfigurationsbits löschen
temp |= 0x33333333; // PB15-PB8 als Push Pull Output definieren
GPIOB->CRH = temp;
}
```

Frage: Aus welchem Grund wird hier eine Variable tmp verwendet ?

```

/********************* U N T E R P R O G R A M M: set_leds ********************/
/* Aufgabe: gibt hexadezimalen Wert auf 8 Leds (PB8-PB15) aus */
/* Input:   value = Wert auf den LEDS gesetzt werden sollen */
/* return: */
/********************* U N T E R P R O G R A M M: get_switches ********************/
/* Aufgabe: liest aktuelle Schalterstellung (PA0-PA7) ein */
/* Input: */
/* return: aktuelle Schalterstellung */
/********************* U N T E R P R O G R A M M: WAIT_MS ********************/
/* Aufgabe: Wartet die angegebene Anzahl an Millisekunden */
/* Input:   ms = Anzahl der zu wartenden Millisekunden */
/* return: */
/********************* W A R T E S C H L E I F E ********************/
void wait_ms(int ms){

int i,j;

for(i = 0; i < ms; i++) {
    for(j = 0; j < 1595; j++) {
    }
}
}

/********************* W A R T E S C H L E I F E ********************/
void wait(){

unsigned char speed;
unsigned char i;

    speed = get_switches();
    speed = ~speed;
    speed = speed & 0x7f;
    for(i= 0; i <= speed; i++){
        wait_ms(100);
    }
}

```

```
/********************************************/  
/*  
     MAIN function  
/********************************************/  
int main (void) {  
  
    char i,hs;lauflicht;  
  
    init_leds_switches();  
    hs = get_switches() & 0x80;  
    while (hs) {  
        lauflicht = 0x01;  
        i = 0;  
        while (i < 8){  
            set_leds(lauflicht);  
            lauflicht = lauflicht <<1;  
            i++;  
            wait();  
        }  
        hs = get_switches() & 0x80;  
    }  
    do{} while (1);  
}
```

1.5 Beispielprogramm #2 – Flanke Detektion, Ausgabe Variable UART

```
/*********************************************
/* (C) Copyright HTL - HOLLABRUNN 2009-2011 All rights reserved. AUSTRIA */
/*
/* File Name: Detect.c
/* Autor: Josef Reisinger
/* Version: V1.00
/* Date: 11/30/2009
/* Description: Erkennen von Flanken, Ausgabe eine HEX Zahl
/*********************************************
/* History: V1.00 creation
/*********************************************
#include <stm32f10x.h>

/*-----Prototypes -----
void uart_init(unsigned long);
void uart_put_char(char);
void uart_put_string(char *);
void uart_put_hex(char);
void nib2asc(char *);
void init_port(void);

/*----- Bit Banding Alias Adresses -----
#define GPIOA_IDR GPIOA_BASE + 2*sizeof(uint32_t)
#define BITBAND_PERI(a,b) ((PERIPH_BB_BASE + (a-PERIPH_BASE)*32 + (b*4)))
#define SW0 (*((volatile unsigned long *)(BITBAND_PERI(GPIOA_IDR, 0))));

/*********************************************
/*          U N T E R P R O G R A M M:
/*
/* Aufgabe: Initialisiert UART1
/* Input: Baudrate
/* return:
/*********************************************
void uart_init(unsigned long baudrate)
{
    RCC->APB2ENR |= RCC_APB2ENR_IOPAEN; //GPIOA mit einem Takt versorgen

    GPIOA->CRH &= ~ (0x00F0);           //loesche PA.9 configuration-bits
    GPIOA->CRH |= (0x0BUL << 4); //Tx (PA9) - alt. out push-pull

    GPIOA->CRH &= ~ (0x0F00); //loesche PA.10 configuration-bits
    GPIOA->CRH |= (0x04UL << 8); //Rx (PA10) - floating
    RCC->APB2ENR |= RCC_APB2ENR_USART1EN; //USART1 mit einem Takt versorgen

    USART1->BRR = 8000000L/baudrate; //Baudrate festlegen

    USART1->CR1 |= (USART_CR1_RE | USART_CR1_TE); //aktiviere RX, TX
    USART1->CR1 |= USART_CR1_UE;                   //aktiviere USART
}
```

```
/****************************************/
/*          U N T E R P R O G R A M M:      */
/**/
/* Aufgabe: Ausgabe eines Zeichens auf UART1      */
/* Input:   ch: Zeichen in ASCII Code      */
/* return:      */
/****************************************/
void uart_put_char(char ch)
{
    while (! (USART1->SR & USART_SR_TXE)); //Data Transmit Register leer?
    USART1->DR = (ch&0xFF); //byteweise reinschieben
}

/****************************************/
/*          U N T E R P R O G R A M M:      */
/**/
/* Aufgabe: Ausgabe eines Strings auf UART1      */
/* Input:   string: C- String der auf UART1 ausgegeben werden soll      */
/* return:      */
/****************************************/
void uart_put_string(char *string)
{
    while (*string) {
        uart_put_char (*string++);
    }
}

/****************************************/
/*          U N T E R P R O G R A M M:      */
/**/
/* Aufgabe: Function converts a Nibble(0-F) to an ASCII ('0'-'F')      */
/* Input:   nib: Nibble to convert      */
/* Output:  nib: Converted Nibble      */
/* return: -      */
/****************************************/
void nib2asc(char *nib)
{
    *nib &= 0x0F; // extract Nibble
    if (*nib >= 0x0A ) { // Hex Value A-F ?
        *nib -= 0x0A;
        *nib += 'A';
    }
    else { // Hex value 0 - 9
        *nib += '0';
    }
}
```

```

/********************* U N T E R P R O G R A M M: *****/
/* Aufgabe: Ausgabe eines 8 Bit Hex Wertes als ASCII String auf UART1 */
/* Input: */
/* return: */
/********************* */

void uart_put_hex(char c)
{
char help;

    help = (c>>4) &0x0f;      // Extract High Nibble
    nib2asc(&help);          // Convert to ASCII Code
    uart_put_char (help);    // Send High Nibble      auf USART1
    help = c&0x0f;            // Extract Low Nibble
    nib2asc(&help);          // Convert to ASCII Code
    uart_put_char (help);    // Send Low Nibble auf USART1
}

/********************* MAIN function *****/
int main (void) {

char zahl;

uart_init(9600);
init_port();
zahl = 0;
do{
    do {                                // Rising edge ?
        }while (SW0 == 0);
    zahl++;
    uart_put_string("Zahl=");
    uart_put_hex(zahl);
    uart_put_string("\xd\xa");
    do{                                // Falling edge ?
        } while (SW0 ==1);
    zahl++;
    uart_put_string("Zahl=");
    uart_put_hex(zahl);
    uart_put_string("\xd\xa");
    } while (1==1);
}

```

1.5.1 Entprellen von Schaltern

Mechanische Schalter wie Drehgeber (Inkrementalgeber) neigen beim Ein- und Ausschalten zum sogenannten **Prellen**, d.h sie schalten schnell mehrfach aus und ein, verursacht durch mechanische Vibrationen des Schaltkontakte, sofern sie nicht dagegen geschützt sind. Vereinfacht dargestellt, sieht eine von einem Schalter oder Taster geschaltete Spannung beim Schalten wie folgt aus:



Soll nun gezählt werden, wie oft ein Kontakt oder ein Relais geschaltet wird, muss das Prellen des Kontaktes exakt berücksichtigt - und von einem gewollten Mehrfachschalten abgegrenzt werden, da sonst möglicherweise Fehlimpulse gezählt- oder andererseits echte Schaltvorgänge übersprungen werden. Für die Vermeidung bzw. Auswertung dieses unsauberen Signals gibt es verschiedene Ansätze

Hardwareentprellung:

- Prellfreie Schalter
- RC – Tiefpass
- RS Flip Flop,

Softwareentprellung:

In den Zeiten der elektronischen Auswertung von Tastern und Schaltern ist das softwaretechnische Entprellen oft billiger, als die Benutzung eines teuren Schalters. Daher werden heute z.B. auch Computertastaturen nicht mehr mit prellarmen Tasten oder Entprellkondensatoren ausgestattet.

Soll nun mit einem Mikrocontroller gezählt werden, wie oft ein Kontakt oder ein Relais geschaltet wird, muss das Prellen des Kontaktes exakt berücksichtigt - und von einem gewollten Mehrfachschalten abgegrenzt werden, da sonst möglicherweise Fehlimpulse gezählt- oder andererseits echte Schaltvorgänge übersprungen werden. Dies muss beim Schreiben des Programms hinsichtlich des Abtastens des Kontaktes unbedingt Rechnung getragen werden.

Die einfachste Methode für dies Problem ist die Definition einer **Entprellzeit**. Sie definiert die Zeit bis der Prellvorgang des Schalters beendet ist. Diese liegt typischerweise bei mechanischen Kontakten zwischen 20 und 40ms. Dies bedeutet das nach der Prellzeit der Kontakt einen stabilen Zustand einnimmt.

Die einfachste Methode zur Realisierung ist die Verwendung einer **Warteschleife**.

Frage: Wie müßte ich Beispielprogramm 2 modifizieren, sodass es auch für einen prellenden Kontakt funktioniert ?

Frage: Welchen Nachteil hat die Warteschleifen Methode – speziell wenn mehere Kontakte zu entprellen sind ?

1.6 Beispielprogramm #3 – Serielle Ausgabe(SoftUart)

```
/*********************************************
/* (C) Copyright HTL - HOLLABRUNN 2009-2011 All rights reserved. AUSTRIA */
/*
/*FileName:SOFTUART.c
/* Autor: JosefReisinger
/*Version:V1.00
/* Date:11/30/2009
/* Description: Serielle Ausgabe über PortPin (SoftUart)
/*********************************************
/* History: V1.00 creation
/*********************************************
#include <stm32f10x.h>

#define GPIOB_ODR GPIOB_BASE + 3*sizeof(uint32_t)
#define BITBAND_PERI(a,b) ((PERIPH_BB_BASE + (a-PERIPH_BASE)*32 + (b*4)))
#define SER_TXD *((volatile unsigned long *) (BITBAND_PERI(GPIOB_ODR,8)))

/*********************************************
/*          W A R T E S C H L E I F E
/*********************************************
void wait() {
char i;
for (i = 0; i<160; i++) {}

/*********************************************
/*          Ausgabe eines Zeichens auf SOFTUART
/*********************************************
void v24_out(char c) {
char i;

    SER_TXD = 0; // Sende Startbit
    wait();
    for (i = 0; i<8; i++) // Sende 8 Datenbit {
        SER_TXD = (c&0x01)?1:0;
        wait();
        c = c>>1; // Nächstes Datenbit
    }
    SER_TXD = 1;
    wait(); // Sende Stopppbit
}

/*********************************************
/*          MAIN function
/*********************************************
int main (void) {

char *t = "DIC - 4ABHEL\n";

    init_leds_switches();
    SER_TXD = 1; // V24 Ruhezustand
    while(*t) { // Ausgabe des gesamten Texts
        v24_out(*t++);
    }
    do {} while(1); // Endlosschleife
}
```

1.7 Beispielprogramm #4 – Ampel

```
/*********************************************
/* (C) Copyright HTL - HOLLABRUNN 2005-2008 All rights reserved. AUSTRIA/
/*********************************************
/*File: Ampel.c                                     */
/*Author: Josef Reisinger                           */
/*Version:1.0                                       */
/*Datum:11/09/2009                                  */
/*Inhalt:Ampelprogramm                            */
/*Hardware:LED/Schalterplatine                   */
/*History:11/09/2009 erstes Release                */
/*********************************************
/*
/*                      AMPELSTEUERUNG
/*
/*          ZEIT UND FARBWAHL IN TABELLENFORM :
/*          BIT BELEGUNG FUER PORT UND TABELLE
/*
/*          | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
/*          -----
/*          SEITE1      | SEITE2      | ZEIT
/*          RT  GB  GN  RT  GB  GN
/*          ZEIT UEBER SCHALTER      0   0
/*          0.5 SEC.           0   1
/*          1.0 SEC.           1   0
/*          1.5 SEC.  TEST      1   1
/*
/*include "armv10_std.h"

/*********************************************
/*          TABELLE-Ampelzustände
/*********************************************
#define Ampelzustaende 20
const char Ampel[] = {
    0x84,          // rot grün      variabel
    0x81,          // ROT FINSTER  0.5S
    0x85,          // ROT GRUEN    0.5S
    0x81,          // ROT FINSTER  0.5S
    0x85,          // ROT GRUEN    0.5S
    0x81,          // ROT FINSTER  0.5S
    0x85,          // ROT GRUEN    0.5S
    0x81,          // ROT FINSTER  0.5S
    0x85,          // ROT GRUEN    0.5S
    0xCA,          // ROT GB GB   1SEC
    0x30,          // GRUEN ROT   VARIABEL
    0x11,          // FINSTER ROT 0.5S
    0x31,          // GRUEN ROT   0.5S
    0x11,          // FINSTER ROT 0.5S
    0x31,          // GRUEN ROT   0.5S
    0x11,          // FINSTER ROT 0.5S
    0x31,          // GRUEN ROT   0.5S
    0x11,          // FINSTER ROT 0.5S
    0x5A,          // GELB ROT GB 1SEC
};

/*********************************************
```

```

/*
          UNTERPROGRAMM: WaitOut
*/
/*Aufgabe: Tabellenparameter fuer Farbe und Zeit wird in Zustand */
/*          Variable uebergeben. UPRO gibt entsprechende Farbe */
/*          mit gewuenschter Zeit auf Peripherie aus */
/*Input: Ampelzustand */
/*Output: keine */
/*Return: keine */
/***************************************************************/
void WaitOut(unsigned char zustand) {

unsigned char zeit,time;

set_leds(zustand & 0xFC);
zeit = zustand & 0x03;
if (zeit == 0) {
    zeit = ~get_switches();
}
for(time = 1;time <= zeit; time++) {
    wait_ms(500);      // Warte 500ms
}
}
/***************************************************************/
/*
          H A U P T P P R O G R A M M
*/
/***************************************************************/
int main (void){

unsigned char zustand_nr;
unsigned char zustand;
char buffer[20];

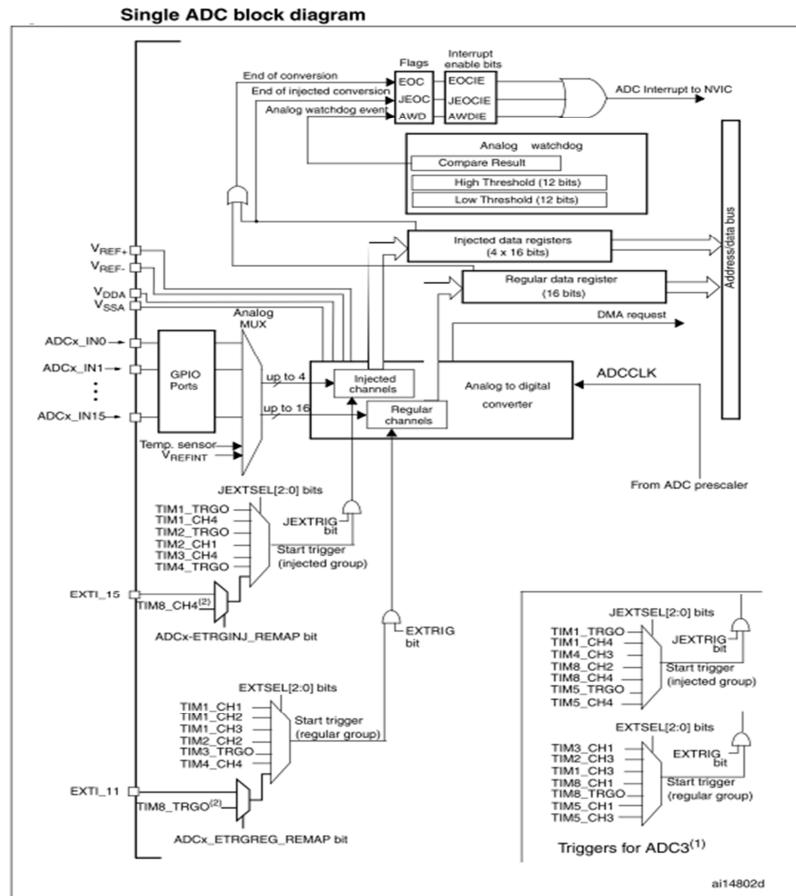
init_leds_switches();      //init Portleitungen für LED Schalterplatine
uart_init(9600);          // 9600,8,n,1
uart_clear();              // Send Clear String to VT 100 Terminal
sprintf(&buffer[0],"CM3-Ampelprogramm:\xd\xa"); //Begrüßung auf V24
uart_put_string(&buffer[0]);
lcd_init();                //LCD initialisieren
lcd_clear();               //LCD löschen
do{
    zustand_nr=0;
    do{
        lcd_set_cursor(0,0); // Cursor auf Ursprung
        sprintf(&buffer[0],"Zustand Nr.= %2d",zustand_nr);
                    // Zustandnummer auf LCD ausgeben
        lcd_put_string(&buffer[0]);
        uart_setpos(0,3);    // Cursor auf Ursprung
        sprintf(&buffer[0],"Zustand Nr.= %2d\xd\xa",zustand_nr);
                    //Zustand auf V24 ausgeben
        uart_put_string(&buffer[0]);
        zustand = Ampel[zustand_nr];
        WaitOut(zustand);
        zustand_nr++;
    }while (zustand_nr < Ampelzustaende);
}while (1==1);
}

```

2 ADC des STM32F10X

2.1 Interner Aufbau

Der STM32F10X enthält 2 12Bit ADC's (ADC1, ADC2), die nach dem Verfahren der sukzessiven Approximation arbeiten. Jeder ADC hat 18 gemultiplexte Kanäle, die es erlauben 16 externe und 2 interne Signale zu messen. Die A/D Conversion der verschiedenen Kanäle kann im "Single", "Continuous", "Scan" oder "discontinuous" mode durchgeführt werden. Das Ergebnis der Konversion (12Bit) kann "left-aligned" oder "right-aligned" im 16 Bit Datenregister abgelegt werden.



1. ADC3 has regular and injected conversion triggers different from those of ADC1 and ADC2.
2. TIM8_CH4 and TIM8_TRGO with their corresponding remap bits exist only in High-density products.

Das **Analog Watchdog** feature erlaubt der Applikation zu detektieren, ob die Eingangsspannungs gewisse Schwellwerte unterschreitet oder überschreitet. Der Input Clock des ADC (PCLK2) mit entsprechendem Vorteiler (Prescaler) darf 14MHz nicht überschreiten.

Die analoge Eingangsspannung an den ADC's kann bei unserem Lehrsystem zwischen 0V(VSSA) und 3,3V (VDDA) liegen. Der ADC wird mit Spannung versorgt bzw aus dem Power Down Modus geweckt sobald das **ADON** Bit im ADC_CR2 Register gesetzt wird.

2.1.1 Kanalauswahl:

Wie bereits vorher erwähnt unterstützt jeder ADC 16 gemultiplexte externe Kanäle. Die Konversionen können in 2 Gruppen - regular und injected eingeteilt werden. Eine Gruppe besteht dabei aus einer beliebigen Folge von Konversionen der verschiedenen Kanäle: z.b: Ch3, Ch8, Ch2, Ch2, Ch0, Ch2, Ch2, Ch15.

Die "regular" Gruppe kann aus seiner Folge von 16 Konversionen bestehen und kann im ADC_SQRx register festgelegt werden. Weiters ist dort die Anzahl der Konversionen (Sequence length) in den L[3:0] Bits festzulegen. Die "injected" Gruppe besteht aus maximal 4 Konversionen und kann im ADC_JSQR register festgelegt werden. Ebenso ist dort in den L[1:0] Bits die Länge der Sequenz festzulegen. Für jeden Kanal kann die Konversionszeit (Anzahl Taktzyklen) individuell im **ADC_SMPRx** Register eingestellt werden. Die minimale Zeit beträgt **1µs**

2.1.2 Single Conversion Mode:

Im Single Conversion Mode wird nur eine einzelne Konversion durchgeführt. Die Konversion wird entweder durch Setzen des **ADON bits** im ADC_CR2 register (nur für regular Kanäle) oder durch einen externen Trigger (für regular und injected Kanäle) gestartet. Das CONT Bit muß dabei auf "0" gesetzt sein. Sobald die Konversion für regular Channels beendet ist wird das Ergebnis im **ADC_DR** register (16Bit) gespeichert und das **EOC Bit** (End of Conversion) gesetzt und optional eine Interrupt (EOCIE) ausgelöst. Das Ergebnis für die injected Channels wird im **ADC_DRJ1** Register gespeichert und das **JEOC Flag** (End Of Conversion Injected) wird gesetzt und optional ein Interrupt (JEOCIE) ausgelöst. Der ADC wird anschließend gestoppt.

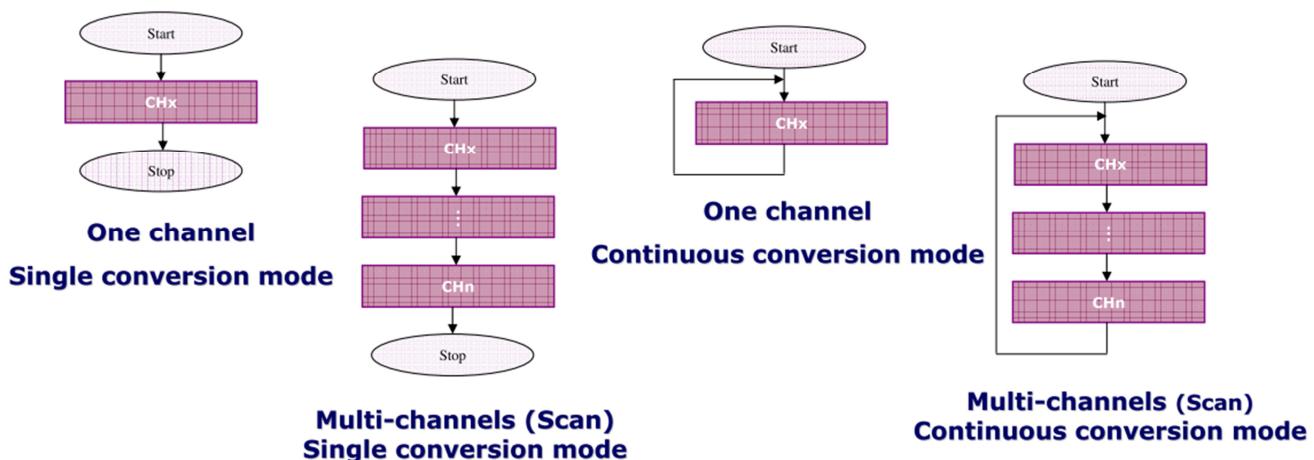
2.1.3 Continuous Conversion Mode:

Im Continuous Mode startet der ADC eine neuerliche Konversion sobald die erste abgeschlossen ist. Dieser Modus kann durch Setzen des **ADON bits** im ADC_CR2 register oder durch einen externen Trigger gestartet werden. Das CONT Bit muß dabei auf "1" gesetzt sein.

Sobald die Konversion für regular Channels beendet ist wird das Ergebnis im **ADC_DR** register (16Bit) gespeichert und das **EOC Bit** (End of Conversion) gesetzt und optional eine Interrupt ausgelöst wenn das EOCIE Bit gesetzt ist. Das Ergebnis für die injected Channels wird im **ADC_DRJ1** Register gespeichert und das **JEOC Flag** (End Of Conversion Injected) wird gesetzt und optional ein Interrupt ausgelöst wenn das JEBCIE Bit gesetzt ist

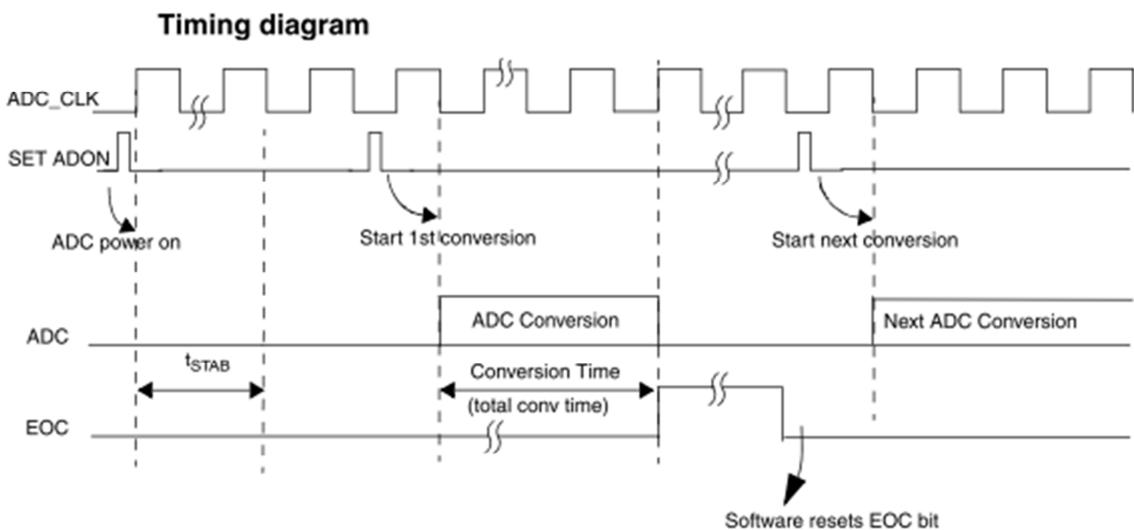
2.1.4 Scan Mode:

Dieser Mode erlaubt eine Folge von Kanälen zu konvertieren. Er wird durch Setzen des SCAN Bits im ADC_CR1 register ausgewählt. Sobald dieses Bit gesetzt ist werden alle Kanäle, die im ADC_SQRx registers (für regular channels) oder im ADC_JSQR (für injected channels) definiert sind, konvertiert. Nachdem ein Kanal konvertiert wurde wird automatisch mit der Konversion des nächsten begonnen. Wenn auch das CONT Bit im ADC_CR2 Register gesetzt wird die Konversion der Folge kontinuierlich durchgeführt.



2.1.5 Timing Diagramm

Wie man aus dem Timing Diagramm erkennt benötigt der ADC eine Stabilisierungszeit t_{Stab} nach dem Einschalten bevor er mit der Konversion beginnen kann. Nach dem Start einer Konversion benötigt er eine Zeit von **14 Taktzyklen** bis er fertig ist und das EOC Flag gesetzt wird und die Daten im ADC_DR Datenregister gespeichert wird.



Nähere Details über den ADC und über andere Betriebsarten wie Discontinuous Mode, Injected Channel Management, Dual Mode bzw dem Analogen Watchdog findet man im Kapitel 11 „Analog to Digital Converter“ des Hardware Reference Manual des STM32F10X

2.1.6 ADC Register

ADC_CR1: ADC Control Register 1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Reserved						AWDEN	JAWDEN	Reserved		DUALMOD[3:0]						
Res.				rw		rw		Res.		rw		rw		rw		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
DISCNUM[2:0]			JDISCE N	DISC EN	JAUTO	AWD SGL	SCAN	JEOC IE	AWDIE	EOCIE	AWDCH[4:0]					
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit 8 SCAN: Scan mode

This bit is set and cleared by software to enable/disable Scan mode. In Scan mode, the inputs selected through the ADC_SQRx or ADC_JSQRx registers are converted.

0: Scan mode disabled

1: Scan mode enabled

Note: An EOC or JEOC interrupt is generated only on the end of conversion of the last channel if the corresponding EOCIE or JEOCIE bit is set

ADC_CR2: ADC Control Register 2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								TSVRE FE	SWST ART	JSWST ART	EXTT RIG	EXTSEL[2:0]			Res.
Res.								rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JEXTT RIG	JEXTSEL[2:0]			ALIGN	Reserved	DMA	Reserved				RST CAL	CAL	CONT	ADON	
rw	rw	rw	rw	rw	Res.	rw	Res.				rw	rw	rw	rw	

Bit 22 SWSTART: Start conversion of regular channels

This bit is set by software to start conversion and cleared by hardware as soon as conversion starts. It starts a conversion of a group of regular channels if SWSTART is selected as trigger event by the EXTSEL[2:0] bits.

0: Reset state

1: Starts conversion of regular channels

Bits 19:17 EXTSEL[2:0]: External event select for regular group

These bits select the external event used to trigger the start of conversion of a regular group:

For ADC1 and ADC2, the assigned triggers are:

000: Timer 1 CC1 event

001: Timer 1 CC2 event

010: Timer 1 CC3 event

011: Timer 2 CC2 event

100: Timer 3 TRGO event

101: Timer 4 CC4 event

110: EXTI line11/TIM8_TRGO event (TIM8_TRGO is available only in high-density devices)

111: SWSTART

Bit 11 ALIGN: Data alignment

This bit is set and cleared by software. Refer to [Figure 30](#).and [Figure 31](#).

0: Right Alignment

1: Left Alignment

Bit 1 CONT: Continuous conversion

This bit is set and cleared by software. If set conversion takes place continuously till this bit is reset.

0: Single conversion mode

1: Continuous conversion mode

Bit 0 ADON: A/D converter ON / OFF

This bit is set and cleared by software. If this bit holds a value of zero and a 1 is written to it then it wakes up the ADC from Power Down state.

Conversion starts when this bit holds a value of 1 and a 1 is written to it. The application should allow a delay of t_{STAB} between power up and start of conversion. Refer to [Figure 26](#).

0: Disable ADC conversion/calibration and go to power down mode.

1: Enable ADC and to start conversion

Note: If any other bit in this register apart from ADON is changed at the same time, then conversion is not triggered. This is to prevent triggering an erroneous conversion.

ADC_SR: ADC Status Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										STRT	JSTRT	JEOC	EOC	AWD	
Res.										rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	

Bit 2 JEOC: Injected channel end of conversion

This bit is set by hardware at the end of all injected group channel conversion. It is cleared by software.

0: Conversion is not complete

1: Conversion complete

Bit 1 EOC: End of conversion

This bit is set by hardware at the end of a group channel conversion (regular or injected). It is cleared by software or by reading the ADC_DR.

0: Conversion is not complete

1: Conversion complete

ADC_SQR1: ADC Regular Sequence Register 1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								L[3:0]				SQ16[4:1]			
Res.								rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ16_0	SQ15[4:0]				SQ14[4:0]				SQ13[4:0]						
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 Reserved, must be kept cleared.

Bits 23:20 **L[3:0]:** Regular channel sequence length

These bits are written by software to define the total number of conversions in the regular channel conversion sequence.

0000: 1 conversion

0001: 2 conversions

....

1111: 16 conversions

Bits 19:15 **SQ16[4:0]:** 16th conversion in regular sequence

These bits are written by software with the channel number (0..17) assigned as the 16th in the conversion sequence.

Bits 14:10 **SQ15[4:0]:** 15th conversion in regular sequence

Bits 9:5 **SQ14[4:0]:** 14th conversion in regular sequence

Bits 4:0 **SQ13[4:0]:** 13th conversion in regular sequence

ADC_SQR2: ADC Regular Sequence Register 2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		SQ12[4:0]					SQ11[4:0]					SQ10[4:1]			
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ10_0	SQ9[4:0]				SQ8[4:0]				SQ7[4:0]						
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30 Reserved, must be kept cleared.

Bits 29:26 **SQ12[4:0]:** 12th conversion in regular sequence

These bits are written by software with the channel number (0..17) assigned as the 12th in the sequence to be converted.

Bits 24:20 **SQ11[4:0]:** 11th conversion in regular sequence

Bits 19:15 **SQ10[4:0]:** 10th conversion in regular sequence

Bits 14:10 **SQ9[4:0]:** 9th conversion in regular sequence

Bits 9:5 **SQ8[4:0]:** 8th conversion in regular sequence

Bits 4:0 **SQ7[4:0]:** 7th conversion in regular sequence

ADC_SQR3: ADC Regular Sequence Register 3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		SQ6[4:0]					SQ5[4:0]					SQ4[4:1]			
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ4_0	SQ3[4:0]				SQ2[4:0]				SQ1[4:0]						
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30 Reserved, must be kept cleared.

Bits 29:25 **SQ6[4:0]:** 6th conversion in regular sequence

These bits are written by software with the channel number (0..17) assigned as the 6th in the sequence to be converted.

Bits 24:20 **SQ5[4:0]:** 5th conversion in regular sequence

Bits 19:15 **SQ4[4:0]:** 4th conversion in regular sequence

Bits 14:10 **SQ3[4:0]:** 3rd conversion in regular sequence

Bits 9:5 **SQ2[4:0]:** 2nd conversion in regular sequence

Bits 4:0 **SQ1[4:0]:** 1st conversion in regular sequence

2.2 Beispielprogramm #5 – Einlesen eines Analogwerts über ADC1

```
/*
 * (C) Copyright HTL - HOLLABRUNN 2009-2011 All rights reserved. AUSTRIA */
/* File Name: std_adc.c */
/* Autor: Josef Reisinger */
/* Version: V1.00 */
/* Date: 24/05/2010 */
/* Description: Einlesen eines Analogwerts über ADC */
/* History: V1.00 creation */
#include "armv10_std.h"
/*
 * MAIN function
*/
int main(void){
char buffer[20];unsigned short int adc_value;
int AD_scaled_ex = 0;int AD_scaled;unsigned char channel;

lcd_init();lcd_clear();
uart_init(9600);
do {
    adc_value=adc_convert();           // Convert Channel 9 of ADC1
    AD_scaled=adc_value/52;          //AD value scaled to 0-78 (lines on LCD)
    if (AD_scaled != AD_scaled_ex){ //If new AD value different than old
        AD_scaled_ex = AD_scaled;
        lcd_set_cursor(0,0);           //1.Zeile 1 (0-1) Position 0 (0-15)
        sprintf(&buffer[0],"ADC Ch%d=%x", channel,adc_value);
        lcd_put_string(&buffer[0]);
        lcd_bargraphXY(1,0,AD_scaled); //Display bargraph according to AD
        sprintf(&buffer[0], "ADC Ch%d=0x%04X\r\n", channel, adc_value);
        uart_put_string(&buffer[0]);
    }
} while(1);
}

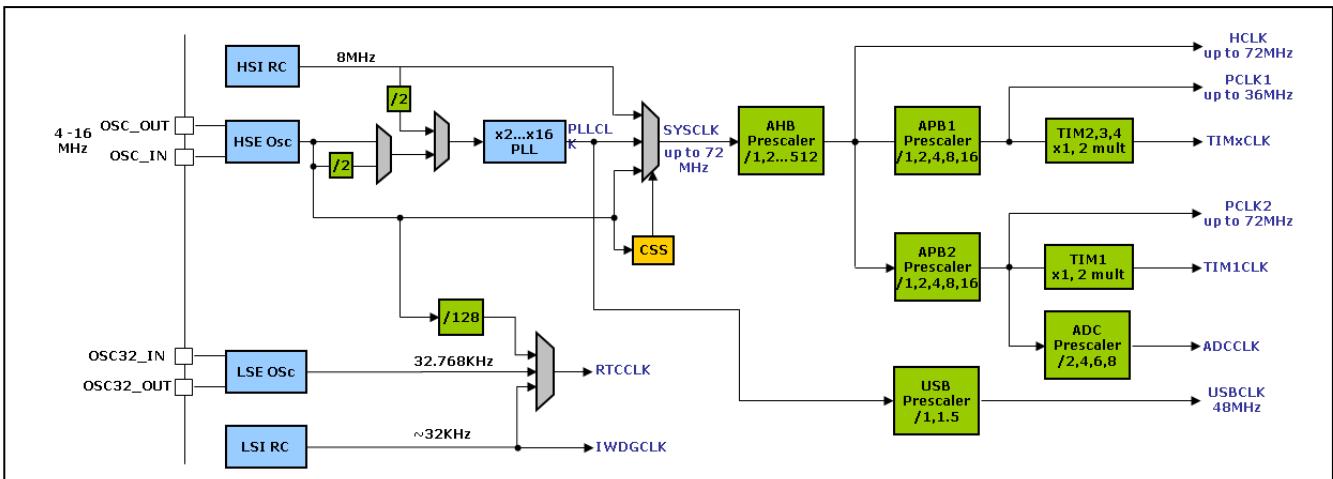
/*
 * U N T E R P R O G R A M M:      adc_convert
 */
/* Aufgabe: delivers value of Portline PB1 (ADC1 Kanal 9) */
/* Input:none */
/* return: converted value (12Bit right aligned) */
unsigned short int adc_convert() {

RCC->APB2ENR |= (1<<9);           // enable periperal clock for ADC1
GPIOB->CRL &= ~0x000000F0;         // set PB1 as analog input
ADC1->CR2 = 0x000E0001;            // right data alignment, SWSTART Conv.
                                         // enable ADC (ADON), Single Mode
ADC1->SMPR2=0x28000000;           //set sample time ADC1 channel 9 (55,5 cycles)
ADC1->SQR1=0x00000000;             //only one conversion item in regular sequence
ADC1->SQR2=0x00000000;
ADC1->SQR3=0x00000000|0x9; //set channel as first in regular conv. sequence
ADC1->CR2|=0x00500000; //start conversion by software for regular sequence
while ((ADC1->SR & 0x2)== 0){}; //Wait till conversion is finished (EOC=1)
return(ADC1->DR);                //deliver converted value of channel
}
```

3 Reset and Clock Control (RCC) des ARM Cortex-M3

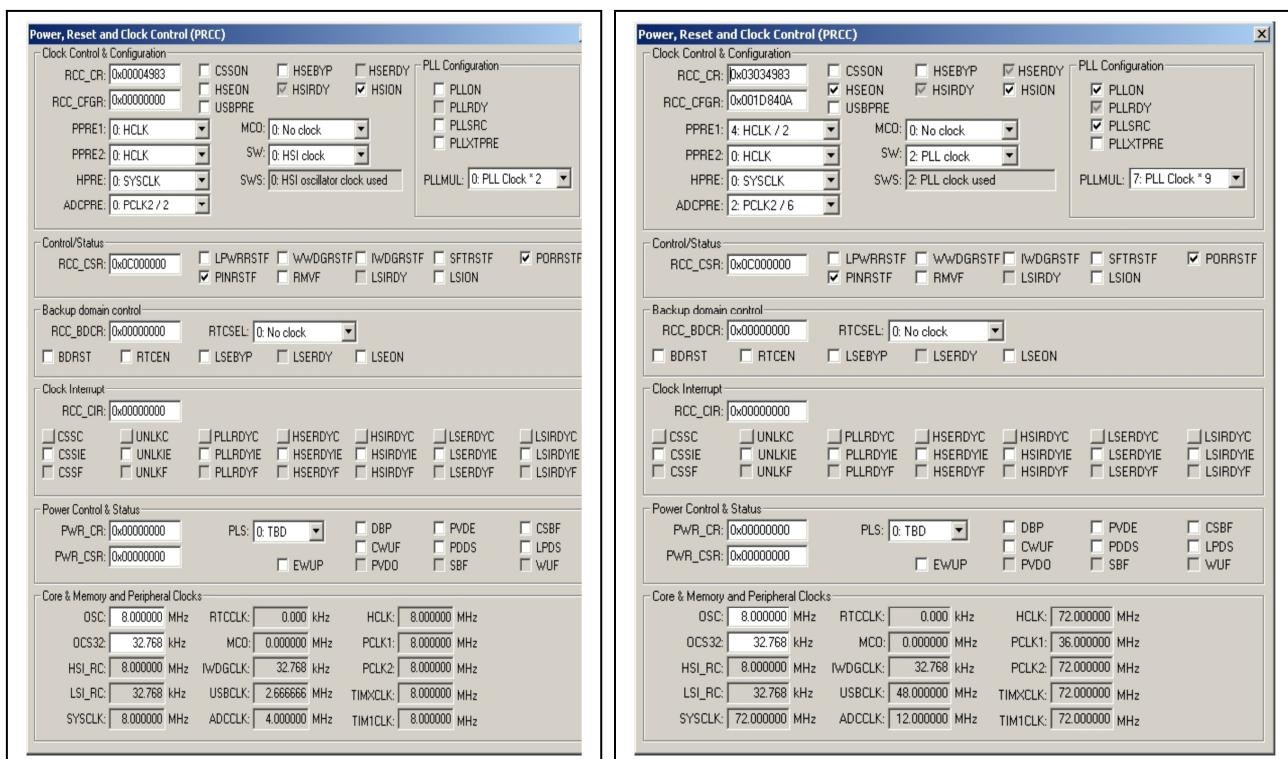
Der Cortex-M3 besitzt 4 mögliche Taktquellen.

- 2 externe Quarze (8MHz und 32,7kHz) und
- 2 interne RC (8Mhz und 40Khz) Oszillatoren.



Standardmäßig wird beim STM32F103RBT6 nach dem Reset der High Speed internal Takt (**HSI**) als System Clock (**SYSCLK**) verwendet. Falls ein externer Quarz oder ein Oszillator angeschlossen ist kann anstatt des HSI auf den externe High Speed Takt (**HSE**) umgeschaltet werden. Fällt diese aus, wird auf den (**HSI**) (8MHz +- 1% getrimmt) umgeschaltet. Die internen PLL kann dazu benutzt werden den HSI oder HSE zu vervielfachen. Durch die **PLL** kann die Frequenz des externen Oszillators (HSE) auf dem Mikrocontroller Board auf bis zu 72MHz angehoben werden. Das heißt, der CPU Takt (also **SYSCLK**) beträgt dann max. 72MHz.

Nun ist es aber so, dass viele Peripherie-Module (unter anderem die Timer) mit dieser max. Taktrate von 72MHz überfordert sind. Daher wird vielen Peripherie-Modulen ein **Prescaler** vorgeschaltet, um den Takt entsprechend anzupassen. Nach dem Reset wird **defaultmäßig HSI mit 8MHz** als SYSCLK verwendet. Im uVision4 Debug Fenster *Peripherals Power, Reset and Clock Control* kann man sich die gerade verwendete Einstellung anzeigen lassen. Im linken Fenster die Einstellung nach dem Reset . Im rechten Fenster ist 72 MHz HSE aktiviert.



Damit die Stromaufnahme nach dem Einschalten niedrig bleibt, muss der Peripherietakt für ADC und die GPIOs in den Registern RCC_APB2ENR und für den ADC CAN USB I2C UASARTs SPI und den Timern im RCC_APB1ENR eingeschaltet werden. Ist der Takt nicht eingeschaltet, können die Peripherieregister nicht ausgelesen werden und liefern immer 00 zurück.

3.1 Beispielprogramm – Set SystemClock

Im folgenden eine Funktion, die den SystemClock auf 72 Mhz hochtaktet

```
void set_clock_72MHz(void) {
    FLASH->ACR = 0x12; //Set Flash latency (2 waitstates)

    RCC->CR |= RCC_CR_HSEON; //HSE on
    while ((RCC->CR & RCC_CR_HSERDY) == 0); // Wait for HSERDY=1 (HSE is ready)

    RCC->CFGR |= RCC_CFGR_PLLMULL9; // 9 mal 8 = 72 MHz (SYSCLK)
    RCC->CFGR |= RCC_CFGR_ADCPRE_DIV6; //ADCCLK = SYSCLK/6 (APB2 PRESCALER=1)
    RCC->CFGR |= RCC_CFGR_PPRE1_DIV2; // PCLK1(APB1)=SYSCLK/2 (HCLK=SYSCLK)
    RCC->CFGR |= RCC_CFGR_PLLSRC; //PLL = SYSCLK, HCLK=SYSCLK, da AHB PRESCALER=1

    RCC->CR |= RCC_CR_PLLON; //PLL on
    while ((RCC->CR & RCC_CR_PLLRDY) == 0); // Wait for PLLRDY=1 (PLL is ready)

    //RCC->CFGR |= RCC_CFGR_SW_HSE; //HSE = Systemclock
    RCC->CFGR |= RCC_CFGR_SW_PLL; //PLL = Systemclock
    while ((RCC->CFGR & RCC_CFGR_SWS_PLL) == 0);
    /* Wait till SYSCLK is stabilized (depending on selected clock) */

    while ((RCC->CFGR & RCC_CFGR_SWS) != ((RCC->CFGR<<2) & RCC_CFGR_SWS));
    // end of stm32_ClockSetup

    RCC->BDCR |= RCC_BDCR_LSEON; //32kHz für RTC siehe AN2821 Reference Manual
        448ff/1072

}
```

- Der HSE (High-Speed-External-Oscillator) wird initialisiert (8MHz bei ARM Minimalsystem)
- Anschließend wird überprüft, ob HSE stabil ist (HSERDY=1)
- Im nächsten Schritte werden die Busse konfiguriert:
PCLK2, welcher die APB2Peripherieeinheiten (ADC1, ADC2, TIM1, UART1, SPI1, GPIOA-E, EXTI, AFIO) mit dem Takt versorgt, hat 72MHz.
PCLK1, welcher Takt für die APB1- Peripherieeinheiten (I2C1, I2C2, SPI2, SPI3, UART2-5, TIM2-7, RTC, WDG,CAN, USB, DAC) ist, hat 36MHz.
- Schließlich muss die PLL noch konfiguriert werden (= HSE*9) und der Systemclock auf PLL gesetzt werden
- Dann muss noch überprüft werden, ob der SystemClock stabil ist (System Clock Switch Status = PLL)
- Als letzter Schritt noch im Backup Domain Controll Register (BDCR) der externe Low Speed Osciallator enabled (LSE). Dieser versort die Realtime Clock (RTC) mit einem Taktsignal.

3.2 Register Clock Control

RCC_CR: Clock Control Register

Reset value: 0x0000 XX83 where X is undefined.

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved						PLL RDY	PLLON	Reserved				CSS ON	HSE BYP	HSE RDY	HSE ON
						r	rw					rw	rw	r	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HSICAL[7:0]								HSITRIM[4:0]				Res.	HSI RDY	HSION	
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw	r	rw	

Bit 25 PLLRDY: PLL clock ready flag

Set by hardware to indicate that the PLL is locked.

0: PLL unlocked

1: PLL locked

Bit 24 PLLON: PLL enable

Set and cleared by software to enable PLL.

Cleared by hardware when entering Stop or Standby mode. This bit can not be reset if the PLL clock is used as system clock or is selected to become the system clock.

0: PLL OFF

1: PLL ON

Bit 17 HSERDY: External high-speed clock ready flag

Set by hardware to indicate that the external 4-25 MHz oscillator is stable. This bit needs 6 cycles of external 4-25 MHz oscillator clock to fall down after HSEON reset.

0: external 4-25 MHz oscillator not ready

1: external 4-25 MHz oscillator ready

Bit 16 HSEON: External high-speed clock enable

Set and cleared by software.

Cleared by hardware to stop the external 1-25MHz oscillator when entering in Stop or Standby mode. This bit cannot be reset if the external 4-25 MHz oscillator is used directly or indirectly as the system clock or is selected to become the system clock.

0: HSE oscillator OFF

1: HSE oscillator ON

Bit 1 HSIRDY: Internal high-speed clock ready flag

Set by hardware to indicate that internal 8 MHz RC oscillator is stable. After the HSION bit is cleared, HSIRDY goes low after 6 internal 8 MHz RC oscillator clock cycles.

0: internal 8 MHz RC oscillator not ready

1: internal 8 MHz RC oscillator ready

Bit 0 HSION: Internal high-speed clock enable

Set and cleared by software.

Set by hardware to force the internal 8 MHz RC oscillator ON when leaving Stop or Standby mode or in case of failure of the external 4-25 MHz oscillator used directly or indirectly as system clock. This bit cannot be reset if the internal 8 MHz RC is used directly or indirectly as system clock or is selected to become the system clock.

0: internal 8 MHz RC oscillator OFF

1: internal 8 MHz RC oscillator ON

RCC_CFGR: Clock Configuration Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved						MCO[2:0]		Res.	USB PRE	PLLMUL[3:0]				PLL XTPRE	PLL SRC
						rw	rw		rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC PRE[1:0]		PPRE2[2:0]			PPRE1[2:0]			HPRE[3:0]				SWS[1:0]	SW[1:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	r	rw	rw

Bits 1:0 SW: System clock switch

Set and cleared by software to select SYCLK source.

Set by hardware to force HSI selection when leaving Stop and Standby mode or in case of failure of the HSE oscillator used directly or indirectly as system clock (if the Clock Security System is enabled).

00: HSI selected as system clock

01: HSE selected as system clock

10: PLL selected as system clock

11: not allowed

Bits 3:2 SWS: System clock switch status

Set and cleared by hardware to indicate which clock source is used as system clock.

- 00: HSI oscillator used as system clock
- 01: HSE oscillator used as system clock
- 10: PLL used as system clock
- 11: not applicable

Bits 14:14 ADCPRE: ADC prescaler

Set and cleared by software to select the frequency of the clock to the ADCs.

- 00: PLCK2 divided by 2
- 01: PLCK2 divided by 4
- 10: PLCK2 divided by 6
- 11: PLCK2 divided by 8

Bits 13:11 PPREG2: APB high-speed prescaler (APB2)

Set and cleared by software to control the division factor of the APB high-speed clock (PCLK2).

- 0xx: HCLK not divided
- 100: HCLK divided by 2
- 101: HCLK divided by 4
- 110: HCLK divided by 8
- 111: HCLK divided by 16

Bits 10:8 PPREG1: APB low-speed prescaler (APB1)

Set and cleared by software to control the division factor of the APB low-speed clock (PCLK1).

Warning: the software has to set correctly these bits to not exceed 36 MHz on this domain.

- 0xx: HCLK not divided
- 100: HCLK divided by 2
- 101: HCLK divided by 4
- 110: HCLK divided by 8
- 111: HCLK divided by 16

Bits 7:4 HPRE: AHB prescaler

Set and cleared by software to control the division factor of the AHB clock.

- 0xxx: SYSCLK not divided
- 1000: SYSCLK divided by 2
- 1001: SYSCLK divided by 4
- 1010: SYSCLK divided by 8
- 1011: SYSCLK divided by 16
- 1100: SYSCLK divided by 64
- 1101: SYSCLK divided by 128
- 1110: SYSCLK divided by 256
- 1111: SYSCLK divided by 512

Note: The prefetch buffer must be kept on when using a prescaler different from 1 on the AHB clock. Refer to [Reading the Flash memory on page 47](#) section for more details.

Bit 16 PLLSRC: PLL entry clock source

Set and cleared by software to select PLL clock source. This bit can be written only when PLL is disabled.

- 0: HSI oscillator clock / 2 selected as PLL input clock
- 1: HSE oscillator clock selected as PLL input clock

Bits 21:18 PLLMUL: PLL multiplication factor

These bits are written by software to define the PLL multiplication factor. These bits can be written only when PLL is disabled.

Caution: The PLL output frequency must not exceed 72 MHz.

- 0000: PLL input clock x 2
- 0001: PLL input clock x 3
- 0010: PLL input clock x 4
- 0011: PLL input clock x 5
- 0100: PLL input clock x 6
- 0101: PLL input clock x 7
- 0110: PLL input clock x 8
- 0111: PLL input clock x 9
- 1000: PLL input clock x 10
- 1001: PLL input clock x 11
- 1010: PLL input clock x 12
- 1011: PLL input clock x 13
- 1100: PLL input clock x 14
- 1101: PLL input clock x 15
- 1110: PLL input clock x 16
- 1111: PLL input clock x 16

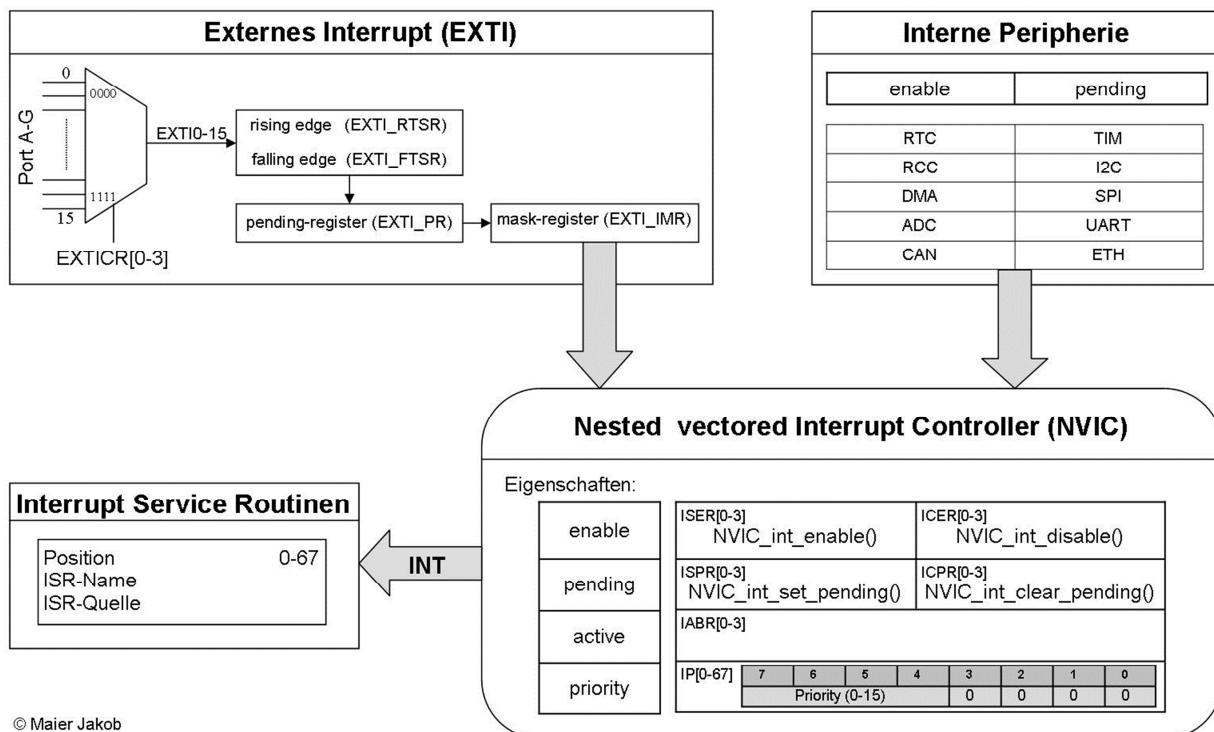
4 Interrupt Programmierung des ARM Cortex-M3

4.1 Einführung

Der STM32F103RB kennt 68 verschiedene Interrupts. Die Unterscheidung erfolgt anhand einer vordefinierten Nummer. Die Interrupt-Service-Routinen Namen sind ebenfalls fix vorgegeben.

Ein Interrupt muss an 2 Stellen konfiguriert werden. Einerseits in der Peripherie, die den Interrupt erzeugt (z.B. Externes Interrupt EXTI, USART, I²C,...) und im **Nested vectored interrupt controller (NVIC)**.

4.2 Blockschaltbild



Wenn eine interne Peripherie oder der Externe Interrupt Controller (EXTI) einen Interrupt an den NVIC sendet, löst dieser bei richtiger Konfiguration einen Interrupt aus und das Programm verzweigt in die Interrupt Service Routine (ISR) entsprechend der Interrupt **Vektortabelle**. Ein Interrupt kann auch manuell ausgelöst werden.

4.3 NVIC Konfiguration

Eine Interrupt kann im NVIC 4 mögliche Zustände haben:

- **enable** Gibt an ob der Interrupt durchgelassen wird
- **pending** Gibt an ob der Interrupt darauf wartet ausgelöst zu werden
- **active** Gibt an ob die ISR gerade ausgeführt wird
- **priority** Ein 4bit Wert der die Priorität des Interrupts angibt (0-15), wobei 0 die höchste und 15 die niedrigste Priorität ist

Mit folgender Funktion NVIC_init() kann die Initialisierung eines Interrupts im NVIC realisiert werden. Position ist die **Interruptnummer** die enabled werden soll (0-67). Mit Priority wird die Interruptpriorität festgelegt. Dies ist ein 4bit-Wert (0-15).

Wenn ein Interrupt an den NVIC gelangt wird immer das **Pending-Bit** gesetzt, egal ob der Interrupt enabled ist oder nicht. Wenn er enabled (**ISER Register**) ist wird dann automatisch in die Interrupt Service Routine gesprungen und das **Active-Bit** gesetzt. Wenn das Programm bereits in einer ISR ist kommt es auf die Interruptpriorität an ob der neue Interrupt hinten angehängt wird oder die aktuelle ISR unterbrochen wird.

Sobald die ISR wieder verlassen wird, wird das Pending-Bit des NVIC automatisch gelöscht um zu verhindern dass diese erneut durchlaufen wird.

Sollte es zu mehreren gleichen Interrupts kommen während der erste noch aktiv ist (Pending=1) werden die anderen nicht mehr ausgeführt und übersprungen.

Der Interrupt kann auch „manuell“ ausgelöst werden, in dem das Pending Bit gesetzt (Pending=1) wird. Das Pending-Bit sollte immer gelöscht werden, bevor ein Interrupt enabled wird, da es sonst möglich ist, das aufgrund eines vergangenen Ereignisses in die ISR gesprungen wird

Der Zugriff auf die Register des NVIC Set-Enable (**ISER**), Clear-Enable (**ICER**), Set-Pending (**ISPR**), Clear-Pending (**ICPR**) and Active Bit(**IABR**) erfolgt entsprechend dem CMSIS Standard über Arrays von 32 Bit Integer Werten entsprechend folgender Tabelle.

Table 41. Mapping of interrupts to the interrupt variables

Interrupts	CMSIS array elements ⁽¹⁾				
	Set-enable	Clear-enable	Set-pending	Clear-pending	Active Bit
0-31	ISER[0]	ICER[0]	ISPR[0]	ICPR[0]	IABR[0]
32-63	ISER[1]	ICER[1]	ISPR[1]	ICPR[1]	IABR[1]
64-67	ISER[2]	ICER[2]	ISPR[2]	ICPR[2]	IABR[2]

1. Each array element corresponds to a single NVIC register, for example the element ICER[1] corresponds to the ICER1 register.

- Das Array ISER[0] to ISER[2] korrespondiert mit den Registern ISER0-ISER2 des Cores
- Das Array ICER[0] to ICER[2] korrespondiert mit den Registern ICER0-ICER2 des Cores
- Das Array ISPR[0] to ISPR[2] korrespondiert mit den Registern ISPR0-ISPR2 des Cores
- Das Array ICPR[0] to ICPR[2] korrespondiert mit den Registern ICPR0-ICPR2 des Cores
- Das Array IABR[0] to IABR[2] corresponds mit den Registern IABR0-IABR2 des Cores

Die Interrupt Prriorität wird über ein Array von 8 Bit Integer Zahlen abgebildet IP[0] to IP[67] und korrespondiert mit den IPR0-IPR67 des Cores. IP[n] enthält die interrupt priority für Interrupt n.

Die Zugriffe auf die Register ist thread safe und erlaubt einen Zugriff auf die Priorität

```
/*
 *          NVIC_init(char position, char priority)
 */
/*Funktion:*/
/*Übernimmt die vollständige Initialisierung eines Interrupts im Nested*/
/*vectored Interrupt controller (Priorität setzen, Auslösen verhindern,*/
/*Interruptenablen)*/
/*Übergabeparameter: "position" = 0-67 (Nummer des Interrupts)*/
/*"priority": 0-15 (Priorität des Interrupts)*/
static void NVIC_init(char position, char priority)
{
    NVIC->ICPR[position >> 0x05] |= (0x01 << (position & 0x1F));
    //Interrupt Clear Pending Register: Verhindert, dass der
    //Interrupt auslöst sobald er enabled wird
    NVIC->IP[position]=(priority<<4);    //Interrupt priority register:
    //Setzen der Interrupt Priorität
    NVIC->ISER[position >> 0x05] |= (0x01 << (position & 0x1F));
    //Interrupt Set Enable Register: Enable interrupt
}
```

4.4 Beispielprogramm #6 – Interrupt UART

```
#include <armv10_std.h>
/*********************************************************/
/*          Uart1 Interrupt Service Routine             */
/*********************************************************/
void USART1_IRQHandler(void)           //ISR
{
    uart_put_char(uart_get_char());    //Zeichen auslesen
    return;
    //pending-bit wird automatisch durch Lesen des UART Datenregister
    (USART1->DR) zurückgesetzt
}

/*********************************************************/
/*          MAIN function                            */
/*********************************************************/
int main (void) {
    uart_init(9600);
    uart_put_string("ready...\n\r");
    NVIC_init(USART1_IRQn,2);      //NVIC: USART1 Global Interrupt Enable
    USART1->CR1|=0x0020;          //USART1 RxNE - Interrupt Enable
    for(;;);
}
```

Hier wird zuerst der USART1 normal initialisiert, anschließend wird der Global Interrupt Enable des Uarts im NVIC erlaubt. Die Priorität ist 2. Zum Schluss wird noch in der USART-Peripherie das Rx-Interrupt-Enable bit (im CR1 Register) gesetzt. Von diesem Zeitpunkt an können Interrupts auftreten. In der ISR wird das Zeichen empfangen und wieder zurückgesendet. Beim empfangen wird das pending-Bit des USARTs automatisch gelöscht, das pending-bit des NVIC wird durch das return gelöscht.

Die Interruptnummern (Positionen) für die einzelnen Interrupts in CMSIS File STM32F01X.h definiert. Der Konstante USART1_IRQn entspricht der Interruptnummer 37. Dies bedeutet der Aufruf könnte auch lauten `NVIC_init(37, 2)`; Der Programmcode ist in diesem Fall natürlich nicht so leicht lesbar.

4.5 Interruptnummern

In dieser Liste stehen alle möglichen Interrupts inklusive deren Nummer:

Position	Priority	Type of priority	Acronym	Description	Address
-	-	-	Reserved		0x0000_0000
-3	fixed	Reset	Reset		0x0000_0004
-2	fixed	NMI	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.		0x0000_0008
-1	fixed	HardFault	All class of fault		0x0000_000C
0	settable	MemManage	Memory management		0x0000_0010
1	settable	BusFault	Pre-fetch fault, memory access fault		0x0000_0014
2	settable	UsageFault	Undefined instruction or illegal state		0x0000_0018
-	-	-	Reserved		0x0000_001C - 0x0000_002B
3	settable	SVCall	System service call via SWI instruction		0x0000_002C
4	settable	Debug Monitor	Debug Monitor		0x0000_0030
-	-	-	Reserved		0x0000_0034
5	settable	PendSV	Pendable request for system service		0x0000_0038
6	settable	SysTick	System tick timer		0x0000_003C
0	7	settable	WWDG	Window Watchdog interrupt	0x0000_0040
1	8	settable	PVD	PVD through EXTI Line detection interrupt	0x0000_0044
2	9	settable	TAMPER	Tamper interrupt	0x0000_0048
3	10	settable	RTC	RTC global interrupt	0x0000_004C
4	11	settable	FLASH	Flash global interrupt	0x0000_0050
5	12	settable	RCC	RCC global interrupt	0x0000_0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000_0064
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000_0068
11	18	settable	DMA1_Channel1	DMA1 Channel1 global interrupt	0x0000_006C
12	19	settable	DMA1_Channel2	DMA1 Channel2 global interrupt	0x0000_0070

19	26	settable	CAN1_TX	CAN1 TX interrupts	0x0000_008C
20	27	settable	CAN1_RX0	CAN1 RX0 interrupts	0x0000_0090
21	28	settable	CAN1_RX1	CAN1 RX1 interrupt	0x0000_0094
22	29	settable	CAN1_SCE	CAN1 SCE interrupt	0x0000_0098
23	30	settable	EXTI9_5	EXTI Line[9:5] interrupts	0x0000_009C
24	31	settable	TIM1_BRK	TIM1 Break interrupt	0x0000_00A0
25	32	settable	TIM1_UP	TIM1 Update interrupt	0x0000_00A4
26	33	settable	TIM1_TRG_COM	TIM1 Trigger and Commutation interrupts	0x0000_00A8
27	34	settable	TIM1_CC	TIM1 Capture Compare interrupt	0x0000_00AC
28	35	settable	TIM2	TIM2 global interrupt	0x0000_00B0
29	36	settable	TIM3	TIM3 global interrupt	0x0000_00B4
30	37	settable	TIM4	TIM4 global interrupt	0x0000_00B8
31	38	settable	I2C1_EV	I ² C1 event interrupt	0x0000_00BC
32	39	settable	I2C1_ER	I ² C1 error interrupt	0x0000_00C0
33	40	settable	I2C2_EV	I ² C2 event interrupt	0x0000_00C4
34	41	settable	I2C2_ER	I ² C2 error interrupt	0x0000_00C8
13	20	settable	DMA1_Channel3	DMA1 Channel3 global interrupt	0x0000_0074
14	21	settable	DMA1_Channel4	DMA1 Channel4 global interrupt	0x0000_0078
15	22	settable	DMA1_Channel5	DMA1 Channel5 global interrupt	0x0000_007C
16	23	settable	DMA1_Channel6	DMA1 Channel6 global interrupt	0x0000_0080
17	24	settable	DMA1_Channel7	DMA1 Channel7 global interrupt	0x0000_0084
18	25	settable	ADC1_2	ADC1 and ADC2 global interrupt	0x0000_0088
35	42	settable	SPI1	SPI1 global interrupt	0x0000_00CC
36	43	settable	SPI2	SPI2 global interrupt	0x0000_00D0
37	44	settable	USART1	USART1 global interrupt	0x0000_00D4
38	45	settable	USART2	USART2 global interrupt	0x0000_00D8
39	46	settable	USART3	USART3 global interrupt	0x0000_00DC
40	47	settable	EXTI15_10	EXTI Line[15:10] interrupts	0x0000_00E0
41	48	settable	RTCAlarm	RTC alarm through EXTI line interrupt	0x0000_00E4
42	49	settable	OTG_FS_WKUP	USB On-The-Go FS Wakeup through EXTI line interrupt	0x0000_00E8
-	-	-	-	Reserved	0x0000_00EC - 0x0000_0104
50	57	settable	TIM5	TIM5 global interrupt	0x0000_0108
51	58	settable	SPI3	SPI3 global interrupt	0x0000_010C
52	59	settable	UART4	UART4 global interrupt	0x0000_0110

53	60	settable	UART5	UART5 global interrupt	0x0000_0114
54	61	settable	TIM6	TIM6 global interrupt	0x0000_0118
55	62	settable	TIM7	TIM7 global interrupt	0x0000_011C
56	63	settable	DMA2_Channel1	DMA2 Channel1 global interrupt	0x0000_0120
57	64	settable	DMA2_Channel2	DMA2 Channel2 global interrupt	0x0000_0124
58	65	settable	DMA2_Channel3	DMA2 Channel3 global interrupt	0x0000_0128
59	66	settable	DMA2_Channel4	DMA2 Channel4 global interrupt	0x0000_012C
60	67	settable	DMA2_Channel5	DMA2 Channel5 global interrupt	0x0000_0130
61	68	settable	ETH	Ethernet global interrupt	0x0000_0134
62	69	settable	ETH_WKUP	Ethernet Wakeup through EXTI line interrupt	0x0000_0138
63	70	settable	CAN2_TX	CAN2 TX interrupts	0x0000_013C
64	71	settable	CAN2_RX0	CAN2 RX0 interrupts	0x0000_0140
65	72	settable	CAN2_RX1	CAN2 RX1 interrupt	0x0000_0144
66	73	settable	CAN2_SCE	CAN2 SCE interrupt	0x0000_0148
67	74	settable	OTG_FS	USB On The Go FS global interrupt	0x0000_014C

4.6 Namen der Interrupt Service Routinen

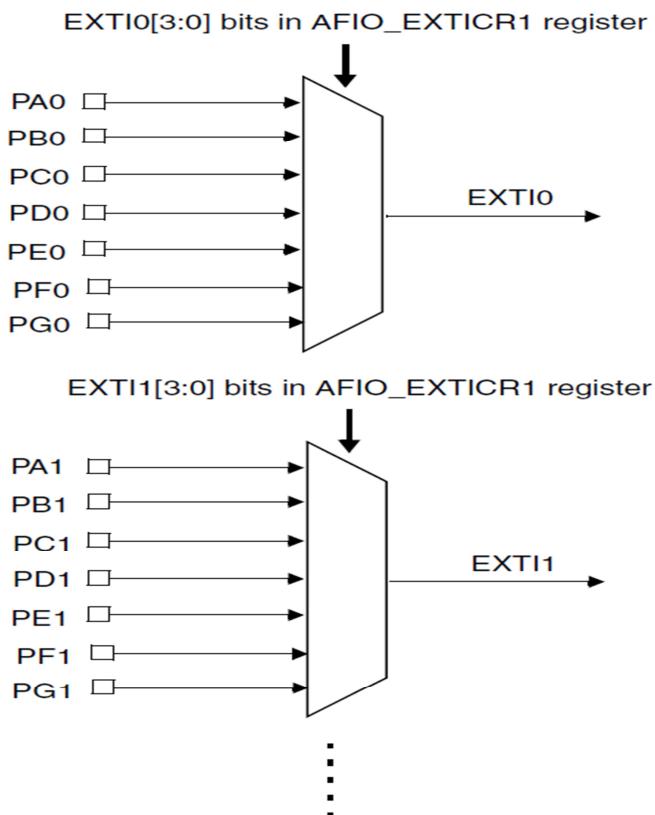
Im Startup Code STM32F10X.s sind entsprechend der Interrupt Vektortabelle alle Einsprungspunkte der entsprechenden Interrupt Service Routinen definiert. Hier eine Liste aller vordefinierten Namen

ISR-Name	Beschreibung
Reset_Handler	Reset Handler
NMI_Handler	NMI Handler
HardFault_Handler	Hard Fault Handler
MemManage_Handler	MPU Fault Handler
BusFault_Handler	Bus Fault Handler
UsageFault_Handler	Usage Fault Handler
SVC_Handler	SVCall Handler
DebugMon_Handler	Debug Monitor Handler
PendSV_Handler	PendSV Handler
SysTick_Handler	SysTick Handler
WWDG_IRQHandler	Window Watchdog
PVD_IRQHandler	PVD through EXTI Line detect
TAMPER_IRQHandler	Tamper
RTC_IRQHandler	RTC
FLASH_IRQHandler	Flash
RCC_IRQHandler	RCC
EXTI0_IRQHandler	EXTI Line 0
EXTI1_IRQHandler	EXTI Line 1
EXTI2_IRQHandler	EXTI Line 2
EXTI3_IRQHandler	EXTI Line 3
EXTI4_IRQHandler	EXTI Line 4
DMAChannel1_IRQHandler	DMA Channel 1
DMAChannel2_IRQHandler	DMA Channel 2
DMAChannel3_IRQHandler	DMA Channel 3
DMAChannel4_IRQHandler	DMA Channel 4
DMAChannel5_IRQHandler	DMA Channel 5
DMAChannel6_IRQHandler	DMA Channel 6
DMAChannel7_IRQHandler	DMA Channel 7

ISR-Name	Beschreibung
ADC_IRQHandler	ADC
USB_HP_CAN_TX_IRQHandler	USB High Priority or CAN TX
USB_LP_CAN_RX0_IRQHandler	USB Low Priority or CAN RX0
CAN_RX1_IRQHandler	CAN RX1
CAN_SCE_IRQHandler	CAN SCE
EXTI9_5_IRQHandler	EXTI Line 9..5
TIM1_BRK_IRQHandler	TIM1 Break
TIM1_UP_IRQHandler	TIM1 Update
TIM1_TRG_COM_IRQHandler	TIM1 Trigger and Commutation
TIM1_CC_IRQHandler	TIM1 Capture Compare
TIM2_IRQHandler	TIM2
TIM3_IRQHandler	TIM3
TIM4_IRQHandler	TIM4
I2C1_EV_IRQHandler	I2C1 Event
I2C1_ER_IRQHandler	I2C1 Error
I2C2_EV_IRQHandler	I2C2 Event
I2C2_ER_IRQHandler	I2C2 Error
SPI1_IRQHandler	SPI1
SPI2_IRQHandler	SPI2
USART1_IRQHandler	USART1
USART2_IRQHandler	USART2
USART3_IRQHandler	USART3
EXTI15_10_IRQHandler	EXTI Line 15..10
RTCAlarm_IRQHandler	RTC Alarm through EXTI Line
USBWakeUp_IRQHandler	USB Wakeup from suspend

4.7 Externe Interrupts

Der Cortex kann jeden Portpin als Interruptquelle initialisieren, auf Grund von Einschränkungen können aber nur max. 16 Externe Interrupts gleichzeitig verwendet werden. Um die Abhängigkeiten zu veranschaulichen hier ein Blockschaltbild:



Es gibt 16 mögliche Interrupt-Lines (EXTI0 – EXTI15) die als **externe Interruptquelle** verwendet werden können. Jede dieser Interrupt-Lines kann mit genau einem Portpin verbunden werden. EXTI0 z.B. aber nur mit PA0, PB0, PC0 usw.. Es gilt folgende Regel: EXTI x = Px n . x ist ein beliebiger Port (A, B, C,...), n ein einzelner Portpin (0-15). Bsp.: PD5 und PE0 nicht gleichzeitig auslösen können. Ein Interrupt kann bei einer positiven und/oder negativen Flanke ausgelöst werden.

4.8 Beispielprogramm #7 – Externer Interrupt

Portpin PA1 erzeugt hier bei jeder fallenden Flanke ein Interrupt. Vor den enable muss das pending-Bit gelöscht werden um zu verhindern dass vergangene Ereignisse am Portpin sofort ein Interrupt auslösen.

```
/*
 * (C) Copyright HTL - HOLLABRUNN 2009-2009 All rights reserved. AUSTRIA */
/*
/* File Name: External_Interrupt.c */
/* Autor: Josef Reisinger */
/* Version: V1.00 */
/* Date: 11/03/2012 */
/* Description: Demoprogramm für External Interrupt */
/*
/* History: V1.00 creation */
/*
#include "armv10_std.h"

/*----- Function Prototypes -----*/
static void NVIC_init(char position, char priority);
static void EXTI1_Config(void);

/*----- Static Variables-----*/
int falling_edges;

/*
 * External Interrupt Service Routine Schalter1 */
void EXTI1_IRQHandler(void) { //ISR
    //Pending bit EXTI1 rücksetzen (Sonst wird die ISR immer wiederholt)
    EXTI->PR |= (0x01 << 1);
    falling_edges++; // Fallende Flanke an PA1 erkannt
    return;
}

/*
 * EXTI1_config
 * Leitung PA1 wird mit EXTI1 verbunden, Interrupt bei falling edge, Priorität 3 */
static void EXTI1_Config(){

    NVIC_init(EXTI1 IRQn, 3); //NVIC fuer initialisieren EXTI Linie (Position 7, Priority 3)

    RCC->APB2ENR |= 0x0001; //AFIOEN - Clock enable
    AFIO->EXTICR[0] &= 0xFFFFFFF0F; //Interrupt-Line EXTI1 mit Portpin PA1 verbinden
    EXTI->FTSR |= (0x01 << 1); //Falling Edge Trigger für EXTI1 Aktivieren
    EXTI->RTSR &=~(0x01 << 1); //Rising Edge Trigger für EXTI1 Deaktivieren

    EXTI->PR |= (0x01 << 1); //EXTI_clear_pending: Das Auslösen auf vergangene
                                //Vorgänge nach dem Enabeln verhindern
    EXTI->IMR |= (0x01 << 1); //Enable Interrupt EXTI1-Line. Kann durch den NVIC
                                //jedoch noch maskiert werden
}
```

```

/***********************/
/*          NVIC_init(char position, char priority)           */
/*Funktion:                                         */
/*Übernimmt die vollständige Initialisierung eines Interrupts im Nested   */
/*vectored Interrupt controller (Priorität setzen, Auslösen verhindern,   */
/*Interruptenablen)                                         */
/*Übergabeparameter: "position" = 0-67 (Nummer des Interrupts)           */
/*"priority": 0-15 (Priorität des Interrupts)                         */
/***********************/
static void NVIC_init(char position, char priority)
{
    NVIC->ICPR[position >> 0x05] |= (0x01 << (position & 0x1F));
    //Interrupt Clear Pending Register: Verhindert, dass der
    // Interrupt auslöst sobald er enabled wird
    NVIC->IP[position]=(priority<<4);    //Interrupt priority register:
    //Setzen der Interrupt Priorität
    NVIC->ISER[position >> 0x05] |= (0x01 << (position & 0x1F));
    //Interrupt Set Enable Register: Enable interrupt
}

/***********************/
/*          MAIN function                                     */
/***********************/
int main (void) {

int lcd_falling_edges;
char buffer[30];

/* GPIO Init: Schalter auf PA1 Leitung als Eingang initialisieren*/
RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;      // enable clock for GPIOA
GPIOA->CRL &= 0xFFFFFFF0F;    // set Port Pins PA1 to Pull Up/Down Input mode
GPIOA->CRL |= 0x00000080;
GPIOA->ODR |= 0x0002;    // Output Register für PA1 auf "1" initialisieren

falling_edges=0;      // Zaehler für falling edges initialisieren
lcd_falling_edges=0;
lcd_init ();        // Initialisieren der LCD Anzeige
lcd_clear();        // LCD Anzeige löschen
//zaehler auf LCD aktualisieren
sprintf(&buffer[0],"Fall. Edges=%d", lcd_falling_edges);
lcd_put_string(&buffer[0]);

EXTI1_Config(); // Konfigurieren des Ext: Interrupts für PA1 (Falling Edges)
do{
    if (lcd_falling_edges != falling_edges)    // Anzeigewert geändert?
    {
        lcd_falling_edges = falling_edges;
        lcd_set_cursor(0,0); // Cursor auf Ursprung
        // zaehler auf LCD aktualisieren
        sprintf(&buffer[0],"Fall. Edges=%d", lcd_falling_edges);
        lcd_put_string(&buffer[0]);
    }
}while (1);
}

```

5 Timer des ARM Cortex-M3

5.1 Übersicht

Timer sind im allgemeinen Zählaufsteine, die für verschiedenste Zwecke eingesetzt werden können (Erzeugung von Impulsformen, Messung von Pulsbreiten, Erzeugung genauer Zeiten,...). Der Cortex-M3 stellt folgende verschiedene Timer bereit

System Tick Timer:

Befindet sich im System Core des Cortex-M3. Wird z.B.: bei einem Scheduler eines Echtzeitbetriebssystems zum Umschalten zwischen verschiedenen Tasks verwendet

Watchdog Timer:

Der Cortex-M3 stellt zwei Watchdog Timer bereit (Windowed Watchdog und Independent Watchdog). Das Watchdog Timer Subsystem schützt das System vor Programmabstürzen. Wird dieser Zähler nicht periodisch nachgeladen so wird ein Reset ausgelöst. Watchdog Timer sind auch im Power Down Modus aktiv.

Baudratengenerator:

Wird verwendet um entsprechende Baudaten für UARTs zu erzeugen

Real Timer Clock:

Die Echtzeituhr läuft auch im Power down Modus des Cortex_M3 weiter und kann auch für Wakup Funktionen eingesetzt werden.

Bei den restlichen Standard Zählern handelt es sich um Up- oder Downcounter, die benutzt werden können externe Impulse zu erfassen oder den Systemtakt zu zählen (Zeitgeber, Timer).

Bei einem Zählerüberlauf wird ein Bit gesetzt (Update Event) und löst eventuell einen Interrupt aus. Die Zähler können jederzeit ausgelesen oder gesetzt werden. Eine hardwaremäßige Kaskadierung der Timer ist ebenfalls möglich.

Advanced control Timer 1 & 8:

Besitzt alle Standard features der General Purpose Timer kann aber zusätzlich drei komplementäre output PWM Paare zur Motorsteuerung erzeugen. Maximale erreichbare Zeit Timer1: 38.5 Stunden. Timer 8 ist nur bei STM32F103XX high density devices vorhanden und steht deshalb beim STM32F103RB nicht zur Verfügung.

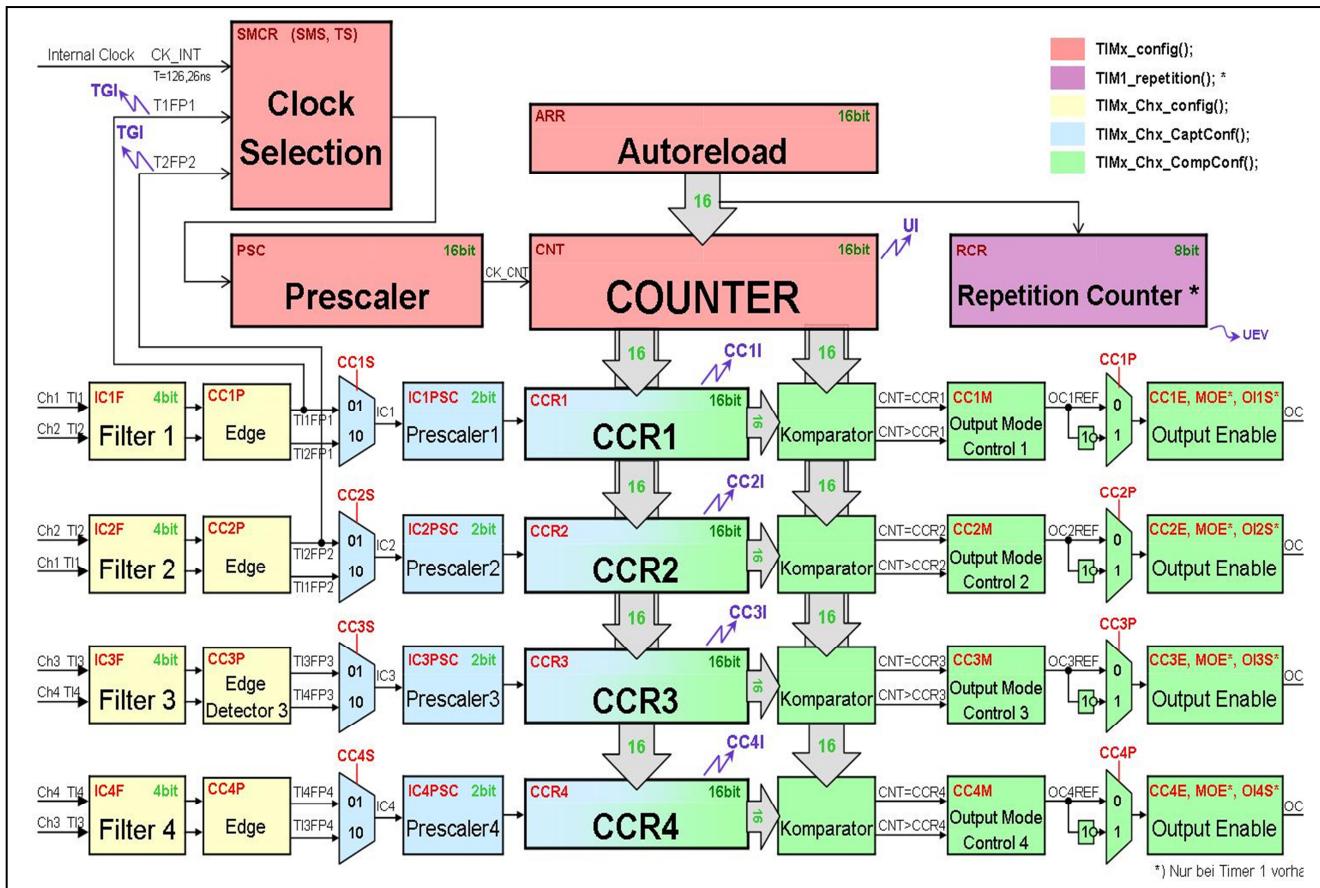
General Purpose Timer 2,3,4,5:

sind kaskadierbare General Purpose Timer mit vier capture/compare Kanälen pro Timer. General Purpose Timer können eine maximale Zeit von 9 min erreichen. Timer 5 steht beim STM32F103RB nicht zur Verfügung. Dieser steht nur bei STM32F103XX high density devices und connectivity line devices zur Verfügung (z.B. STM32F107VC)

Basic Timer: 6&7:

Sind low speed APB1 getaktet mit Synchronisationsausgang für den DAC . Die beiden Timer stehen beim STM32F103RB nicht zur Verfügung. Diese stehen nur bei STM32F103XX high density devices und connectivity line devices zur Verfügung (z.B. STM32F107VC)

5.2 Blockschaltbild eines Timers



Der Hauptblock eines Timers besteht aus folgenden Einheiten

- Clock Selektion (**TIMx_SMCR**) – dient zur Auswahl des Taktsignals
- 16 Bit Zähler (**TIMx_CNT**)
- 16 Bit Prescaler (**TIMx_PSC**)
- 16 Bit Auto-Reload Register (**TIMx_ARR**)

Timer 1 hat zusätzlich noch ein 8bit Repetition Counter Register (**TIM1_RCR**). Es kann jeder Timer unabhängig von anderen Ressourcen programmiert werden.

5.3 Konfiguration eines Timers

5.3.1 Clock Selection

Die Clock Selection kann auf folgende Arten initialisiert werden. Diese kann mit den drei SMS Bits im TIMx Slave mode Control Register (**TIMx_SMCR**) festgelegt werden.

- CK_INT Es wird eine interne Taktquelle verwendet ($f \approx 8\text{MHz}$)
- Channel1 Channel 1 (Externer Port) wird als Takt verwendet
- Channel2 Channel 2 (Externer Port) wird als Takt verwendet
- Ch1_reset CK_INT ist die Taktquelle, Channel 1 macht ein Update-Event (Reset)
- Ch2_reset CK_INT ist die Taktquelle, Channel 2 macht ein Update-Event (Reset)
- Ch1_gated CK_INT ist die Taktquelle, Channel 1 enables den Takt
- Ch2_gated CK_INT ist die Taktquelle, Channel 2 enables den Takt
- Ch1_trigger CK_INT ist die Taktquelle, Channel 1 startet den Counter
- Ch2_trigger CK_INT ist die Taktquelle, Channel 2 startet den Counter

Bei den Taktarten kann man hauptsächlich zwischen dem **Timerbetrieb** (CK_INT) und einem **Counterbetrieb (Channel1, Channel2)** unterscheiden. Allerdings kann der externe Pin noch für weitere Zwecke verwendet werden:

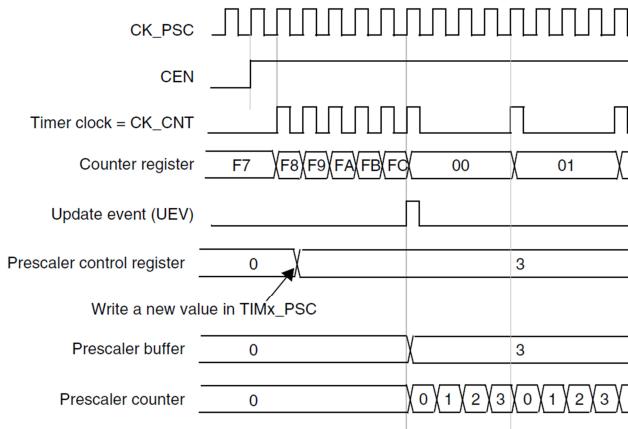
- Eine Flanke lässt den Counter um eines weiter zählen (Modus:"Channel1","Channel2")
- Eine Flanke führt zu einem Reset des Counters (Modus:"Ch1_reset","Ch2_reset")
- Der Pin enables den internen Takt (Modus:"Ch1_gated","Ch2_gated")
- Eine Flanke am Pin startet den Counter (Modus:"Ch1_trigger","Ch2_trigger")

Bis auf "Channel1" und "Channel2" wird bei jedem Modus CK_INT als Taktquelle verwendet. Die Frequenz von CK_INT kann leicht variieren und hat eine Periodendauer von etwa 125ns (8MHz Systemclock). Wenn ein externer Takt verwendet werden soll, muss dazu ein Capture-Compare Channel als Eingang initialisiert werden (Channel 1 oder Channel 2) und der entsprechende Port konfiguriert werden. Für jede Taktquelle kann der 16bit Prescaler verwendet werden.

5.3.2 Prescaler

Die Bitbreite des Prescalers ist bei jedem Timer 16bit. Er dient dazu die Eingangsfrequenz, die von der Clock Selection kommt, durch einen bestimmten Faktor zu teilen und an den Counter weiterzuleiten.

$$f_{\text{OUT}} = f_{\text{IN}} / \text{PSC}$$



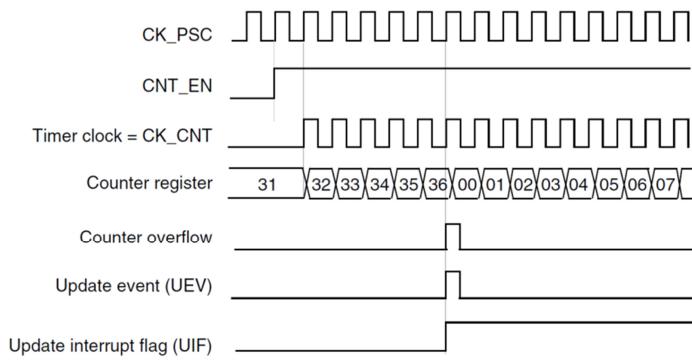
Dieses Beispiel zeigt eine Änderung des Prescaler-Wertes von 1 zu 4 während des Betriebes. CK_PSC ist der Eingangstakt der von der Clock Selection kommt. CEN enabled den Counter. CK_CNT ist der Ausgang des Prescalers der in den Counter führt. Man erkennt dass der neue Wert erst übernommen wird, wenn ein Update-Event (UEV) auftritt. Dies kann entweder manuell ausgelöst werden oder automatisch.

Die minimale Frequenz die erreicht werden kann ist bei 7,92MHz 121Hz.
 $(f_{\text{OUT}} = 7,92\text{MHz} / (2^{16}-1) = 121\text{Hz})$

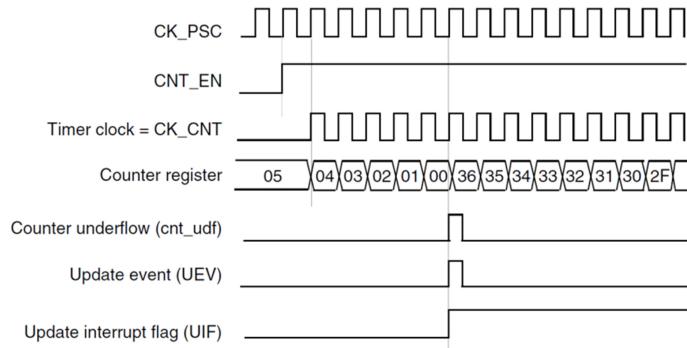
5.3.3 CounterModi

Das Autoreload-Register (TIMx_ARR register) hat eine Bitbreite von 16bit. Der Inhalt gibt den Zählbereich des Counters an. Wenn der Counter z.B. als **Upcounter** im Edge Aligend Mode (DIR Bit = 0 und CMS=00 in TIMX_CR1 Register) betrieben wird zählt der Counter von 0 bis zum Wert des Autoreload-Registers.

In folgenden Beispiel steht im Prescaler-Register 1, das heißt dass CK_PSC = CK_CNT. Im Autoreload-Register steht der Wert 36. Nach dem Überlauf des Counters wird ein **Update-Event** (UEV) erzeugt, was zu einem Rücksetzen des Counters führt. Wenn das **UIF-Flag** im TIMx_DIER register gesetzt wurde, wird auch ein Interrupt erzeugt. Der UEV Event kann durch Setzen des **UDIS Bit** in TIMX_CR1 Register unterdrückt werden

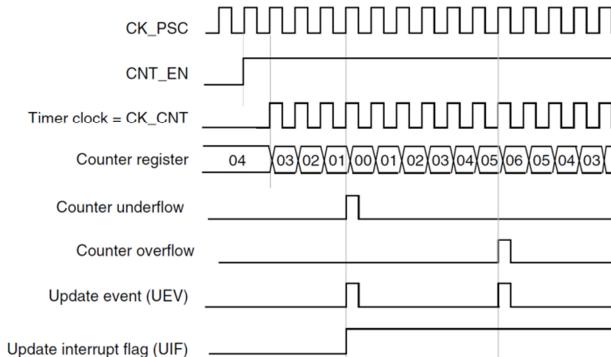


Wenn der Counter als **Downcounter** im Edge Aligend Mode (DIR Bit = 1 und CMS=00) in TIMX_CR1 Register betrieben wird, zählt dieser vom Wert des Autoreload-Registers bis 0 abwärts.



In diesem Beispiel wurden die Register genauso wie im ersten Beispiel gesetzt: Im Prescaler-Register steht 1, im Autoreload-Register 36.

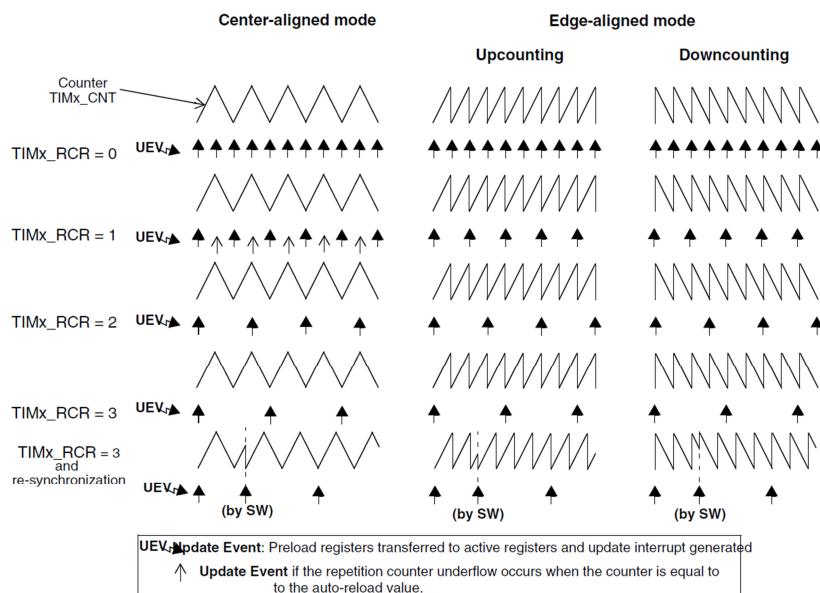
Die 3. Möglichkeit ist der Up- & Down-Modus (**Center Aligned Mode**) bei dem der Counter abwechselnd nach oben (0 bis TIMx_ARR register -1) und anschließend nach unten (TIMx_ARR register bis 1) zählt.



Hier steht im Prescaler wieder 1. Im Autoreload-Register steht 6. Ein Update-Event (UEV) wird sowohl beim Überlauf als auch beim Unterlauf erzeugt. Die **CMS** Bits im TIMX_CR1 Register können auf die Werte 01,02 oder 03 gesetzt werden. Die Modi unterscheiden sich nur im Zusammenhang mit der Verwendung der Output Compare Einheit (Siehe nächstes Kapitel). In diesem Modus kann das **DIR** Bit im TIMx_CR1 register nicht beschrieben werden. Es wird in diesem Fall von der Hardware automatisch gesetzt, um die Richtung anzugeben.

5.3.4 Repetition Counter

Dieses Register ist ein 8bit-Register und nur beim Timer 1 verfügbar. Es wird als eine Art zusätzlicher Prescaler für das Update-Event (UEV) verwendet was im PWM-Modus nützlich sein kann. (PWM-Modus siehe Capture-Compare Channels)



Dieses Diagramm zeigt an, wann ein Update Event (UEV) durchgeführt wird. Ein Update Event bewirkt, dass alle Register der Timer aktualisiert und Änderungen übernommen werden. Die Diagramme symbolisieren den Zählerstand.

5.4 Interrupt Konfiguration der Timer

Mit dem **TIMx_DIER** Register (Timer Interrupt/DMA Enable Register) kann festgelegt werden, welche Interrupts beim jeweiligen Timer ausgelöst werden können.

Folgende Interrupts sind bei einem General Purpose Timer standardmäßig möglich:

- **Trigger_Int** Wenn Ch1 oder Ch2 die Taktquelle sind und auslösen
- **Update_Int** Ein Interrupt wird ausgelöst wenn es zu einem Über- oder Unterlauf kommt

Es können auch mehrere Interrupts gleichzeitig erlaubt werden. Es müssen dann die jeweiligen Bits gesetzt werden. Die Interrupt-Service Routinen haben folgende Namen:

Interrupt-Service Routinen Namen:

- **Timer 1: void TIM1_UP_IRQHandler(void);** (Für das Update-Interrupt)
void TIM1_TRG_COM_IRQHandler(void); (Für das Trigger Interrupt)
- **Timer 2: void TIM2_IRQHandler(void);** (für alle Interrupts)
- **Timer 3: void TIM3_IRQHandler(void);** (für alle Interrupts)
- **Timer 4: void TIM4_IRQHandler(void);** (für alle Interrupts)

In der jeweiligen ISR Routine müssen die entsprechenden Pending-Bits gelöscht werden. Dies muss in der ISR am Beginn geschehen, um zu verhindern dass der entsprechende Interrupt immer wieder auslöst.

5.5 Timer Zeitberechnung

Die resultierende Zeit eines Timers ist immer vom Takt abhängig. Dadurch, dass die Taktfrequenz CK_INT bei jedem Mikrocontroller leicht variiert ist es notwendig diese zuvor zu bestimmen um ein genaues Ergebnis zu erhalten. Die Zählrichtung des Timers spielt bei der Zeitberechnung keine Rolle.

5.5.1 Messung und Berechnung der Taktfrequenz

Bei der Zeitberechnung ist es am Einfachsten einen Timer zu verwenden, der bei einem Überlauf einen Interrupt auslöst. In der ISR wird ein Portpin getoggelt, der mit einem Oszilloskop beobachtet wird.

Bei der Timer-Konfiguration sollte der Prescaler 0 gesetzt werden und im Autoreload-Register ein möglichst großer Wert stehen.

Die Taktfrequenz ergibt sich dann so:

$$f_{CK_INT} = \frac{ARR}{T}$$

T ist die gemessene Zeitspanne des High- bzw. Low-Pegels des Portpins, ARR der Wert im Autoreload-Register. Dieser Vorgang sollte mehrmals mit verschiedenen ARR-Werten wiederholt werden.

Wenn der Prescaler erhöht wird sollte nun folgende Formel zutreffen:

$$f_{CK_INT} = \frac{ARR}{T} * (PSR + 1)$$

Das Ergebnis der Taktfrequenz sollte in etwa die Taktfrequenz des externen Taktes sein. Dieser ist im Normalfall ein 8MHz Quarz.

5.5.2 Zeitberechnung

Die Berechnung erfolgt am Einfachsten unter Verwendung der Periodendauer. Die Periodendauer des internen Takts ist bei 8MHz 125ns.

$$T_{CK_CNT} = T_{CK_INT} * (PSC + 1)$$

T_{CK_CNT} ist die Periodendauer des Signals das in den Counter geführt wird.
Ein Über- bzw. Unterlauf tritt alle ARR Zyklen vor.

Es ergibt sich folgende Formel:

$$T = T_{CK_CNT} * ARR = T_{CK_INT} * (PSC + 1) * ARR$$

An dieser Formel kann man erkennen dass der kleinste Wert der ins Autoreload-Register geschrieben werden kann 1 ist. Der Wert 0 ist verboten, da der Timer in diesem Fall nicht zählen kann.

Anleitung zur Berechnung der Register:

$$ARR = \frac{T_{gesucht}}{T_{CK_INT}}$$

Wenn	$ARR < 65535:$	$PSC = 0$
Sonst:	$ARR = 0xFFFF$	

$$PSC = \frac{T_{gesucht}}{T_{CK_INT} * 65535} - 1$$

Berechnungsbeispiele:

Anmerkung: Die gemessene Periodendauer T_{CK_INT} beträgt 126,26ns.

- $T_{gesucht} = 400ms$
 $ARR = \frac{400ms}{126,26ns} = 3.168.066 \rightarrow ARR = 0xFFFF$
 $PSC = \frac{400ms}{126,26*65535} - 1 = 47 \rightarrow PSC = 47$
- $T_{gesucht} = 6ms$
 $ARR = \frac{6ms}{126,26ns} = 47521 \rightarrow ARR = 47521 \quad PSC = 0$
- $T_{gesucht} = 500\mu s$
 $ARR = \frac{500\mu s}{126,26ns} = 3960 \rightarrow ARR = 3960 \quad PSC = 0$
- $T_{gesucht} = 1s$
 $ARR = \frac{1s}{126,26ns} \rightarrow ARR = 0xFFFF$
 $PSC = \frac{1s}{126,26*65535} - 1 = 120 \rightarrow PSC = 120$

5.6 Beispielprogramm #8 – Lauflicht mit Timer Interrupt

```
*****  
/* (C) Copyright HTL - HOLLABRUNN 2009-2009 All rights reserved. AUSTRIA */  
/*  
/* File Name: Lauflicht_Interrupt.c */  
/* Autor: Josef Reisinger */  
/* Version: V1.00 */  
/* Date: 11/03/2012 */  
/* Description: Demoprogramm für Timer 1 */  
*****  
/* History: V1.00 creation */  
*****  
#include "armv10_std.h"  
  
----- Function Prototypes -----*/  
static void TIM1_Config(void);  
static void NVIC_init(char position, char priority);  
  
----- Static Variables-----*/  
static int sekunden;  
  
*****  
/* Interrupt Service Routine Timer1 (General Purpose Timer) */  
*****  
void TIM1_UP_IRQHandler(void) { //Timer 1, löst alle 1000ms aus  
//Interrupt pending bit löschen (Verhinderung eines nochmaligen Interrupt-auslösens)  
    TIM1->SR &= ~0x01;  
    sekunden++;  
}  
  
*****  
/* Initialization Timer1 (General Purpose Timer) */  
*****  
static void TIM1_Config(void) {  
  
    ----- Timer 1 konfigurieren -----*/  
    RCC->APB2ENR |= 0x0800; //Timer 1 Clock enable  
    TIM1->SMCR = 0x0000; //Timer 1 Clock Selection: CK_INT wird verwendet  
    TIM1->CR1 = 0x0000; //Auswahl des Timer Modus: Upcounter  
    /* T_INT = 126,26ns, Annahme: */  
    /* Presc = 150 -> Auto Reload Wert=52801 (=0xCE41)-> 1s Update Event*/  
    TIM1->PSC = 150; //Wert des prescalers (Taktverminderung)  
    TIM1->ARR = 0xCE41; //Auto-Reload Wert = Maximaler Zaehlerstand des Upcounters  
    TIM1->RCR = 0; //Repetition Counter deaktivieren  
  
    ----- Interrupt Timer 1 konfigurieren -----*/  
    TIM1->DIER = 0x01; //enable Interrupt bei einem UEV (Überlauf / Unterlauf)  
    NVIC_init(TIM1_UP_IRQn, 2); //Enable Timer 1 Update Interrupt, Priority 2  
  
    ----- Timer 1 Starten -----*/  
    TIM1->CR1 |= 0x0001; //Counter-Enable bit setzen  
}
```

```

/*****
/*          NVIC_init(char position, char priority)          */
/*Funktion:                                         */
/*Übernimmt die vollständige Initialisierung eines Interrupts im Nested   */
/*vectored Interrupt controller (Priorität setzen, Auslösen verhindern,   */
/*Interruptenable)                                         */
/*Übergabeparameter: "position" = 0-67 (Nummer des Interrupts)           */
/*                      "priority": 0-15 (Priorität des Interrupts)           */
/*****
static void NVIC_init(char position, char priority)
{
    NVIC->ICPR[position >> 0x05] |= (0x01 << (position & 0x1F));
    //Interrupt Clear Pending Register: Verhindert, dass der
    // Interrupt auslöst sobald er enabled wird
    NVIC->IP[position]=(priority<<4);    //Interrupt priority register:
    //Setzen der Interrupt Priorität
    NVIC->ISER[position >> 0x05] |= (0x01 << (position & 0x1F));
    //Interrupt Set Enable Register: Enable interrupt
}

/*****
/*          MAIN function                                     */
/*****
int main (void) {
int lauflicht;           // Lauflicht P1 Abbild
int lcd_sekunden;        // aktueller Zählerstand auf LCD
char buffer[30];

init_leds_switches();
uart_init(9600);         // 9600,8,n,1
uart_clear();             // Send Clear String to VT 100 Terminal
lcd_init();               // LCD initialisieren
lcd_clear();              // Lösche LCD Anzeige
lauflicht = 0x01;         // Lauflicht im Speicher initialisieren
sekunden=0;                // Sekundenzaehler initialisieren
lcd_sekunden = 0;          // aktueller Anzeigewert
set_leds(0);               // LED löschen
TIM1_Config(); //Timer 1 starten: Upcounter --> löst alle 1s einen Update Interrupt
uart_put_string("Lauflicht mit Interrupt:\n");
do {
    if (lcd_sekunden != sekunden) {      // 1 Sekunde vergangen ?
        lcd_sekunden = sekunden;
        set_leds(lauflicht);            // Lauflicht auf LED aktualisieren
        lauflicht = lauflicht*2;        // nächstes Lauflicht Bit
        if (lauflicht==256){           // letztes Lauflicht Bit ?
            lauflicht = 0x01;          // Anfangszustand Lauflicht
        }
        lcd_set_cursor(0,0);           // Cursor auf Ursprung
        // ----- Sekundenzaehler auf LCD aktualisieren-----
        sprintf(&buffer[0],"Sekunden=%d", lcd_sekunden);
        lcd_put_string(&buffer[0]);
    }
} while (1);
}

```

5.7 Capture Compare Channel

Jeder der Timer besitzt 4 Capture/Compare Kanäle. Diese können unabhängig voneinander als Eingang (Input Capture) oder Ausgang (Output Compare) initialisiert werden. Standardmäßig werden die Kanäle nicht verwendet. Die Kanäle 1 und 2 können, wenn sie als Eingang konfiguriert wurden, auch als Taktquelle verwendet werden. Jeder Kanal ist einem bestimmten Portpin zugeordnet. Dieser kann auch auf einen alternativen Portpin gemapped (remapped) werden.

Standardpins:

	Timer 1	Timer 2	Timer 3	Timer 4
Channel 1	PA8	PA0	PA6	PB6
Channel 2	PA9	PA1	PA7	PB7
Channel 3	PA10	PA2	PB0	PB8
Channel 4	PA11	PA3	PB1	PB9

5.7.1 Interrupt Konfiguration für Capture/ Compare Einheit

Wie bereits im vorgrigen Kapitel erwähnt kann mit dem **TIMx_DIER** Register (Timer Interrupt/DMA Enable Register) festgelegt werden, welche Interrupts beim jeweiligen Timer ausgelöst werden können. Für die den Capture/Compare Kanäle sind noch folgende zusätzliche Interrupts möglich

- CC1_int
 - CC2_int
 - CC3_int
 - CC4_int
- } Es kommt zu einem Interrupt für den jeweiligen Kanal, wenn der Counterwert abgespeichert wird (Input Capture) oder der Counterwert mit dem jeweiligen CCR-Wert übereinstimmt (Output Compare).

Es können auch mehrere Interrupts gleichzeitig erlaubt werden. Es müssen dann nur die jeweiligen Enable Bits gesetzt werden.

Timer 1 hat für den Trigger-Interrupt, die Capture-Compare Interrupts und den Update-Interrupt unterschiedliche Interrupt Service Routinen, alle anderen Timer haben nur eine ISR für alle Interrupts.

Interrupt-Service Routinen Namen:

- Timer 1:
 void TIM1_UP_IRQHandler(void); (Für das Update-Interrupt)
 void TIM1_TRG_COM_IRQHandler(void); (Für das Trigger Interrupt)
 void TIM1_CC_IRQHandler(void); (Für alle Capture/Compare Interrupts)
- Timer 2:
 void TIM2_IRQHandler(void); (für alle Interrupts)
- Timer 3:
 void TIM3_IRQHandler(void); (für alle Interrupts)
- Timer 4:
 void TIM4_IRQHandler(void); (für alle Interrupts)

In der jeweiligen ISR Routine müssen die entsprechenden Pending-Bits gelöscht werden. Dies muss in der ISR geschehen, um zu verhindern dass diese immer wieder auslöst.

5.7.2 Input Capture

Input Capture heißt, dass der gewählte Kanal auf seinem Port horcht bis eine Flanke auftritt. Wenn dies geschieht speichert er den aktuellen Zählerstand des Counters in seinem Capture-Compare-Register ab und erzeugt (wenn er zuvor enabled wurde) einen Interrupt.

Als erster Schritt muss der entsprechende Portpin als Input definiert werden. Beispielweise ist dies Portleitung PA1 bei Timer 2 um Kanal 2 als Input Capture verwenden zu können (CNF1 = 1 CNF1 = 0 Mode1=Mode0=0 ODR=1 für Input Pull Up Mode)

```
GPIOA->CRL &= 0xFFFFF0F;           // PA1 Konfigurationsbits löschen  
GPIOA->CRL |= 0x00000080;         // PA1 als Input definieren  
GPIOA->ODR |= 0x0002;             // PA1 internen Pull Up aktivieren
```

Als nächstes muss der Input Kanal **enabled** werden. Dies geschieht durch Setzen des entsprechenden Bits im Timer Capture/Compare Enable Register **TIMx_CCER**. Weiters kann die Polarität des Channels festgelegt werden (nicht invertiert/ invertiert)

Beispiel: Channel 2 von Timer ist als Input Capture definiert

```
TIM2->CCER |= 0x0020;           //polarity TIM2_CH2 (capture at falling edge)  
TIM2->CCER |= 0x0010;           //Enable Capture2
```

Ebenso kann der ausgewählte Kanal später wieder disabled werden. Dies geschieht durch Löschen des entsprechenden Enable Bits.

```
TIM2->CCER &= 0xFFEF;          //Disable Capture2
```

Im Folgenden muss nun der Kanal als **Eingangsstufe** konfiguriert werden. Zusätzlich kann für die Eingangsstufe ein Filter und ein Prescaler definiert werden. Für Kanal 1 und 2 ist das Capture/Compare Mode Register **TIMx_CCMR1** verantwortlich bzw. **TIMx_CCMR2** für Kanal 3 und Kanal 4. Hier als Beispiel soll die Eingangsstufe 2 von Timer 2 konfiguriert werden.

```
TIM2->CCMR1 |= 0x0100; //CH2 ist Input Capture für Timer2, No prescaler  
TIM2->CCMR1 &= 0x0FFF; //Konfiguriere Input Capture 2 filter für Timer2  
                      // (no filter → sampling at full speed)
```

Der Filter definiert die Frequenz mit der der Eingang abgetastet wird und die Länge des verwendeten digitalen Filters.

0000	No filter, sampling is done at fDTS
0001	fsAMPLING=fCK_INT N=2.
0010	fsAMPLING=fCK_INT N=4
0011	fsAMPLING=fCK_INT N=8
0100	fsAMPLING=fDTS/2 N=6
0101	fsAMPLING=fDTS/2 N=8
0110	fsAMPLING=fDTS/4 N=6
0111	fsAMPLING=fDTS/4 N=8
1000	fsAMPLING=fDTS/8 N=6
1001	fsAMPLING=fDTS/8 N=8
1010	fsAMPLING=fDTS/16 N=5
1011	fsAMPLING=fDTS/16 N=6
1100	fsAMPLING=fDTS/16 N=8
1101	fsAMPLING=fDTS/32 N=5
1110	fsAMPLING=fDTS/32 N=6
1111	fsAMPLING=fDTS/32 N=8

Im Normalfall ist kein Filter notwendig.

Wie dem Blockschaltbild zu entnehmen ist, kann für Capture Unit 1 und 2 wahlweise Kanal 1 oder Kanal 2 als Input verwendet werden. Für Capture Unit 3 und 4 kann wahlweise Kanal 3 oder Kanal 4 als Eingang verwendet werden. Es ist zu beachten dass es nicht möglich ist die Channel1-2 mit den Channeln 3-4 auszukreuzen !

Der Capture Wert wird durch Auslesen des Registers **TIMx_CCRx** abgefragt werden. Beispielweise Kanal2 von Timer 2 erhält man durch Auslesen des Registers TIM2->CCR2

5.8 Beispielprogramm #9 – Timer Capture Demoprogramm

```
*****  
/* (C) Copyright HTL - HOLLABRUNN 2009-2009 All rights reserved. AUSTRIA */  
/*  
/* File Name: Timer_Capture.c  
/* Autor: Josef Reisinger  
/* Version: V1.00  
/* Date: 16/01/2016  
/* Description: Demoprogramm für Capture Eingang an GP Timer 2 */  
*****  
/* History: V1.00 creation */  
*****  
#include "armv10_std.h"  
  
/*----- Function Prototypes -----*/  
static void TIM2_Config(void);  
static void NVIC_init(char position, char priority);  
  
/*----- Static Variables-----*/  
__IO uint16_t IC2ReadValue1 = 0, IC2ReadValue2 = 0;  
__IO uint16_t CaptureNumber = 0;  
__IO uint32_t Capture = 0;  
  
*****  
/* Timer 2 Interrupt Service Routine */  
*****  
void TIM2_IRQHandler(void)  
{  
    if (((TIM2->SR & TIM_IT_Update) != 0) && ((TIM2->DIER&TIM_IT_Update) != 0))  
        // Update Interrupt?  
    {  
        TIM2->SR &= ~TIM_IT_Update; /* Clear TIM2 Update Interrupt pending bit */  
        LED7=~LED7;  
    }  
    if (((TIM2->SR&TIM_IT_CC2) != 0) && ((TIM2->DIER & TIM_IT_CC2) != 0))  
        // Capture Interrupt of Channel 2?  
    {  
        TIM2->SR &= ~TIM_IT_CC2; /*Clear TIM2 Capture Interrupt pending bit forChannel 2*/  
        if(CaptureNumber == 0)  
        {  
            IC2ReadValue1 = TIM2->CCR2; /* Get the Input Capture value for Channel 2*/  
            CaptureNumber = 1;  
        }  
        else if(CaptureNumber == 1)  
        {  
            IC2ReadValue2 = TIM2->CCR2; /* Get the Input Capture value for Channel 2*/  
            if (IC2ReadValue2 > IC2ReadValue1) /* Capture computation */  
            {  
                Capture = (IC2ReadValue2 - IC2ReadValue1);  
            }  
            else  
            {  
                Capture = ((TIM2->ARR - IC2ReadValue1) + IC2ReadValue2);  
            }  
            CaptureNumber = 0;  
        }  
    }  
}  
#define TIM_CounterMode_Up ((uint16_t)0x0000)  
#define TIM_ICSelection_DirectTI ((uint16_t)0x0001)  
#define TIM_ICPolarity_Rising ((uint16_t)0x0000)  
#define TIM_IT_Update ((uint16_t)0x0001)  
#define TIM_IT_CC2 ((uint16_t)0x0004)
```

```
/*
 *          Initialization Timer2 (General Purpose Timer)
 */
static void TIM2_Config(void)
{
    uint16_t tmp = 0;

    /*----- Configure Timer 2 CR1 - Register -----*/
    RCC->APB1ENR |= RCC_APB1Periph_TIM2;           /* Timer 2 Clock enable */
    tmp=TIM2->CR1;
    tmp&=~(TIM_CR1_DIR|TIM_CR1_CMS);             /* CMS=00: Edge-aligned mode: */
    /* The counter counts up or down depending on direction bit. */
    tmp |= TIM_CounterMode_Up;                    /* DIR=0-->Upcounter */
    TIM2->CR1=tmp;
    TIM2->SMCR=0x0000;                          /* SMS=000: Counter is directly clocked by CLK_INT */
    TIM2->PSC = 7936;                           /* Prescaler: reduce system clock rate to 1KHz */
    TIM2->ARR = 1000;                            /*Auto-Reload Value */

    /* - Configure Ch2(PA1)as Input Capture Channel (Capture Compare mode Register 1) */
    tmp=TIM2->CCMR2;
    tmp&= (~TIM_CCMR1_CC2S)&(~TIM_CCMR1_IC2F);   /* full speed, not filter */
    tmp|=TIM_ICSelection_DirectTI<<8;           /*CC2S=01-->01: CC2 channel as input capture,
    /* IC2 is mapped on TI2. Channel 2 is connected to PA1 */
    tmp &= ~TIM_CCMR1_IC2PSC;                   /* IC2PSC= 00: no Input Capture 1 prescaler */
    /* capture is done each time an edge is detected on the capture input */
    TIM2->CCMR1 = tmp;                          /* Write to TIM2 CCMR2 registers */

    /* --- Conf. Capture Compare Enable register for Input Capture Channel 2 -----*/
    TIM2->CCER &= ~TIM_CCER_CC2E;      /* Disable the Channel 2: Reset the CC2E Bit */
    tmp = TIM2->CCER;
    tmp &= ~TIM_CCER_CC2P; /* Set Polarity: non-inverted: capture onrising edge of IC2 */
    tmp|= (TIM_ICPolarity_Rising<<4)|TIM_CCER_CC2E; /* Enable Input Capture Channel 2 */
    TIM2->CCER = tmp;                         /* Write to TIM2 CCER registers */

    /* ----- Enable Interrupts for Timer 2 -----*/
    TIM2->DIER |= TIM_IT_Update;    /* Enable Update Interrupt */
    TIM2->DIER |= TIM_IT_CC2;      /* Enable the CC2 Interrupt Request */
    NVIC_init(TIM2 IRQn,1);        /* Enable Timer 2 Interrupt */
}

/*
 *          MAIN function
 */
int main (void)
{
    char buffer[50];

    lcd_init();                                /* Init LCD Display */
    init_leds_switches(); /* Schalter S0(PA0)- S7(PA7) und LED L0(PB8)- L7(PB15) */
    TIM2_Config();                             /* Configure Timer 2 */

    /*----- Timer 2 Starten -----*/
    TIM2->CR1 |= TIM_CR1_CEN;      /* Enable the TIM2 Counter */
    do {
        wait_ms(100);
        lcd_clear();
        lcd_set_cursor(0,0);
        sprintf (&buffer[0], "CNT=%d", TIM2->CNT);
        lcd_put_string(&buffer[0]);
        lcd_set_cursor(1,0);
        sprintf (&buffer[0], "%d:%d ", IC2ReadValue1, IC2ReadValue2);
        lcd_put_string(&buffer[0]);
    } while (1==1);
}
```

5.8.1 Output Compare

Beim Output-Compare Modus ist der Portpin des Channels als Ausgang definiert. Mithilfe eines Komparators wird der Zählstand des Counters ständig mit dem Wert im Capture-Compare Register (CCR) verglichen der zuvor gesetzt wurde. Wenn die Werte übereinstimmen wird eine bestimmte, zuvor eingestellte Operation am Ausgang durchgeführt.

Zuerst muss der entsprechende Portpin des verwendeten Kanals als **Alternate Function Push Pull** (CNF1 = 1 CNF0 = 0 Mode1=Mode0=1 ODR=don't care) konfiguriert werden. Hier beispielsweise Portpin PB8 für Kanal3 des Timer 4

```
GPIOB->CRH &= 0x0000000F; //Bereich löschen
GPIOB->CRH |= 0x0000000B; //PB8 in Mode Alternate Function Push Pull
                           //definieren =>TIM4_CH3
```

Als nächstes muss der Kanal der Capture/Compare Einheit als **Output Compare** und der **Output mode** definiert werden. Dies geschieht durch Konfigurieren des **TIMx_CCMR2** Registers. Dabei wird mit dem Output Mode festgelegt, was bei einer Übereinstimmung mit dem Ausgang geschehen soll. Es gibt folgende Möglichkeiten:

- | | |
|------------------|--|
| • frozen | Der Ausgang behält seinen Zustand bei |
| • match_active | Sobald der Zählstand übereinstimmt ist der Ausgang immer 1 |
| • match_inactive | Sobald der Zählstand übereinstimmt ist der Ausgang immer 0 |
| • match_toggle | Sobald der Zählstand übereinstimmt wird der Ausgang getoggelt |
| • force_active | Der Ausgang ist 1, unabhängig vom Zählerstand |
| • force_inactive | Der Ausgang ist 0, unabhängig vom Zählerstand |
| • PWM_mode1 | Upcounter: CCR < CounterVal: Ausgang=1, sonst 0
Downcounter: CCR < CounterVal: Ausgang=0, sonst 1 |
| • PWM_mode2 | Upcounter: CCR < CounterVal: Ausgang=0, sonst 1
Downcounter: CCR < CounterVal: Ausgang=1, sonst 0 |

Hier ein Beispielcode um Kanal 3 der Capture/Compare Einheit 3 von Timer 4 im Output PWM Mode 3 zu konfigurieren.

```
TIM4->CCMR2 &= 0xFFFF; //Capture/Compare Kanal 3 als Output definieren
TIM4->CCMR2 |= 0x0068; //Capture/Compare Mode Register:
                        //Capture/Compare Kanal 3->PWM Mode1: Upcounter -->
                        //CNT < CounterVal: Ausgang=1, sonst 0
```

Der **Vergleichswert** kann im Register **TIMx_CCRx** gesetzt werden. Der Vergleichswert ist ein 16 Bit Wert (0x0000 – 0xFFFF). Im Folgenden Beispielcode wird der Vergleichswert für die Capture/Compare Einheit 3 von Timer 4 gesetzt.

```
TIM4->CCR3 = (TIM4->ARR) / 2; //Vergleichswert Capture/Compare 3 laden (Hälfte
                                //des maximalen Zaehlerstands)
```

Schließlich muss die Caputure/Compare Einheit noch enabled (erlaubt) werden. Dies geschieht durch Setzen des entsprechenden Bit im Capture /Compare Enable Register (TIMx_CCER). Im Folgenden soll die Capture/Compare Einheit 3 des Timers 4 enabled werden.

```
TIM4->CCER |= 0x0100; //Enable CCU3 enable --> Ausgang TIM4_CH3 auf 0
```

Diese kann natürlich später durch Löschen des Flags wieder disabled werden. Im folgenden wieder ein Beispielcode für CCU3 des Timers 4.

```
TIM4->CCER &= 0xF0FF; // disable CCU3 von Timer4
```

Wenn im Update Event Register (**TIMX_EGR**) das entsprechende Bit gesetzt ist, wird bei einer Zählerübereinstimmung ein Event generiert. Dieser kann einen Interrupt auslösen falls dieser enabled ist. (**TIMx_DIER**)

```
TIM4->EGR |= 0x0008; // Update Event generated (reinitialize registers)
```

Achtung.

Wenn Channel1 oder 2 als Taktquelle verwendet wird, kann dieser nicht gleichzeitig als Ausgang geschalten werden !!

5.8.2 Countermodi für Output Compare

Der Counter kann wie oben bereits im vorigen Kapitel erwähnt in folgenden Modi betrieben werden. Dieser kann im **TIMx_CR1** Register mit den beiden CMS Bits festgelegt werden:

- **CMS=00-> Edge Aligend Mode:**
Der Counter zählt von 0 bis zum Wert des Autoreload-Registers (UpCounter) oder vom Wert des Autoreload-Registers bis 0 (Down Counter) in Abhängigkeit vom direction Bit DIR.
DIR=0 -> Upcounter, DIR=1-> Downcounter
- **CMS=01->Center Aligned Mode1:**
Der Counter arbeitet als Upcounter oder als Downcounter in Abhängigkeit vom DIR Bit. Der Output Compare Interrupt wird nur beim Hochzählen aktiviert.
- **CMS=10->Center Aligned Mode2:**
Der Counter arbeitet als Upcounter oder als Downcounter in Abhängigkeit vom DIR Bit. Der Output Compare Interrupt wird nur beim Zählen nach unten aktiviert.
- **CMS=11->Center Aligned Mode3:**
Der Counter arbeitet als Upcounter oder als Downcounter in Abhängigkeit vom DIR Bit. Der Output Compare Interrupt wird bei jeder Übereinstimmung aktiviert.
- In **One-pulse-mode** (OPM Flag=1) stoppt der Timer automatisch beim ersten Über-/Unterlauf.

Der Unterschied zwischen den verschiedenen Up-&Downcounter – Modi ist das Erzeugen von Compare-Interrupts. Deshalb sind die unterschiedlichen Konfigurationen nur zu beachten wenn mindestens ein Capture-Compare Channel des Timers als Output konfiguriert ist und der Output-Compare Interrupt enabled wird

5.9 Beispielprogramm #10 –Output Compare mit PWM Demoprogramm

```
*****
/* (C) Copyright HTL - HOLLABRUNN 2009-2009 All rights reserved. AUSTRIA */
/*
/* File Name: Timer_Compare_PWM.c
/* Autor: Josef Reisinger
/* Version: V1.00
/* Date: 31/03/2012
/* Description: LED Dimmer: Demoprogramm für Timer Compare
/* Funktionalitaet (PWM Out)
/*
/* History: V1.00 creation
/*
*****
```

```
#include "armv10_std.h"

----- Function Prototypes -----
static void TIM4_Config(void);
static void NVIC_init(char position, char priority);

----- Static Variables-----
/*****
/* Timer 4 Interrupt Service Routine
/*****
void TIM4_IRQHandler(void){ //Timer 4 = PWM Output Mode, Channel 3
static char direction=1; //Auf- / Abdimmern

if(direction==0) {
    TIM4->CCR3++; //Längere High-Phase > heller
    if(TIM4->CCR3>=(TIM4->ARR/2)) {
        direction=1; //Danach abdimmern
    }
}
else {
    TIM4->CCR3--; //Längere Low-Phase > dünkler
    if(TIM4->CCR3==0) {
        direction=0; //Danach aufdimmen
    }
}
TIM4->SR &=~0x00FF; //Interrupt pending-bit löschen
return;
}

/*****
/* NVIC_init(char position, char priority)
/* Funktion:
/* Übernimmt die vollständige Initialisierung eines Interrupts im Nested
/* vectored Interrupt controller (Priorität setzen, Auslösen verhindern,
/* Interrupt enable)
/* Übergabeparameter: "position" = 0-67 (Nummer des Interrupts)
/* "priority": 0-15 (Priorität des Interrupts)
/*****
static void NVIC_init(char position, char priority)
{
    NVIC->ICPR[position >> 0x05] |= (0x01 << (position & 0x1F));
    //Interrupt Clear Pending Register: Verhindert, dass der
    // Interrupt auslöst sobald er enabled wird
    NVIC->IP[position]=(priority<<4); //Interrupt priority register:
    //Setzen der Interrupt Priorität
    NVIC->ISER[position >> 0x05] |= (0x01 << (position & 0x1F));
    //Interrupt Set Enable Register: Enable interrupt
}
```

```

/********************* Initialization Timer4 (General Purpose Timer) *****/
static void TIM4_Config(void) {

/*----- Timer 4 konfigurieren -----*/
RCC->APB1ENR |= 0x0004; //TIM4 Clock enable
TIM4->SMCR = 0x0000;    //Slave Mode disabled - CK_INT wird verwendet
TIM4->CR1 = 0x0000;    //Upcounter: Zählt von 0 bis zum Wert des
                      //Autoreload-Registers
TIM4->CR1 |= 0x0080;   //ARPE-Bit setzen (Auto Reload Register buffered)

/* T_INT = 126,26ns, Annahme: Presc = 0x00FF --> Auto Reload Wert = 0x00FF (=256)
/* --> 8,27ms */
TIM4->PSC = 0xFF; //Prescaler
TIM4->ARR = 0xFF; //Auto-Reload Register
TIM4->DIER = 0x01; // Enable Update Interrupt
NVIC_init(TIM4 IRQn,1); // Enable Interrupt Timer4 at NVIC (Priority 1)

/*-----Configure Compare Unit Timer 4: PWM Modus 1 auf Channel 3 -----*/
TIM4->CCER &=0xF0FF; //Capture/Compare Enable Register:
                      //Disable Output Capture/Compare3
TIM4->CCR3 = (TIM4->ARR)/2; //Vergleichswert Capture/Compare 3 laden
                      //(Hälfte des max Zaehlerstand)
TIM4->CCMR2 |=0x0068; //Capture/Compare Mode Register:
                      //Capture/Compare Kanal 3->PWM Model: Upcounter -->
                      //CNT < CounterVal: Ausgang=1, sonst 0
TIM4->CCMR2 &=0xFFFF; //Als Capture/Compare Kanal 3 als Output definieren

TIM4->CCER |= 0x0100; //Enable Output Capture/Compare3 -->
                      //Ausgang TIM4_CH3 auf 0
}

/********************* MAIN function *****/
int main (void){

init_leds_switches(); //LED-Schalter Platine initialisieren
                     //(Schalter PA0-PA7, LED PB8-PB15

GPIOB->CRH &=~ 0x0000000F; //Bereich löschen
GPIOB->CRH |= 0x0000000B; //PB8 (=LED0) in Mode Alternate Function Push Pull
                           //definieren => TIM4_CH3

TIM4_Config(); // Timer 4 konfigurieren (PWM Modus auf Kanal 3)
/*----- Timer 4 Starten -----*/
TIM4->CR1 |= 0x0001; //Counter-Enable bit für Timer 4 setzen

do {

} while (1==1);

}

```

6 DMA Controller des ARM Cortex-M3

6.1 Allgemeiner Aufbau

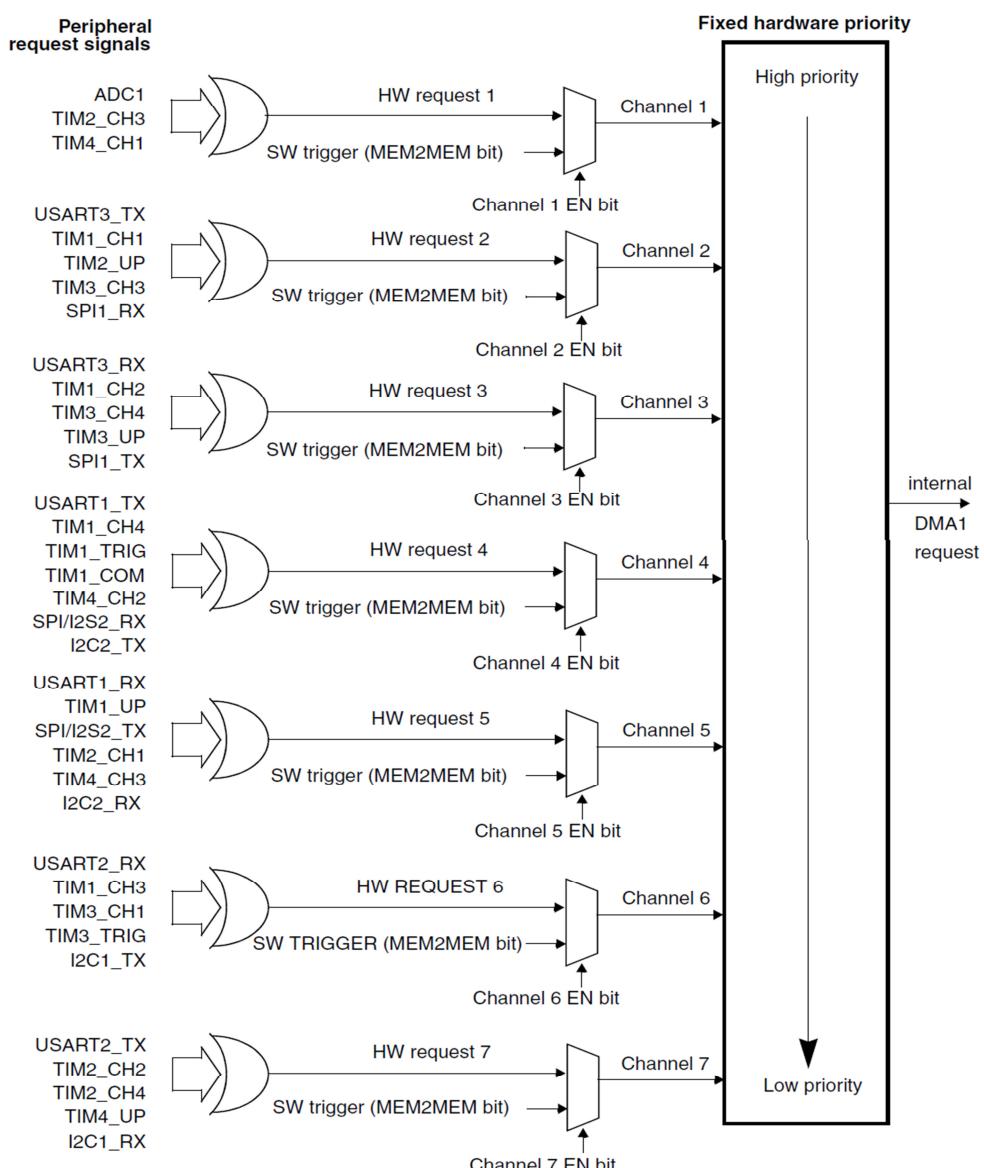
Der DMA (Direct Memory Access) wird verwendet, um sehr schnellen Datentransfer zwischen Peripherie und Speicher oder Speicher und Speicher zu ermöglichen. Der Vorteil von DMA besteht darin, dass zum Datentransfer die CPU nicht benötigt wird und so für andere Zwecke währenddessen zur Verfügung steht.

Der STM32F103RB besitzt zwei DMA Controller (DMA1 bzw. DMA2), die insgesamt 12 Channels (Kanäle) bedienen können. Der DMA1 stellt 7 Kanäle, der DMA2 stellt 5 Kanäle bereit. Jeder DMA Kanal ist bestimmten Peripherieeinheiten zugewiesen. Die DMA Controller werden durch Anfragen (Requests) einer oder mehreren Peripherieeinheiten aktiv. Außerdem können Prioritäten für die einzelnen Requests vergeben werden.

Die DMA Controller ermöglichen folgende Transfers:

- Memory to Memory
- Peripheral to Memory
- Memory to Peripheral
- Peripheral to Peripheral

Aufbau von DMA1:



In folgenden beiden Abbildungen sieht man die Peripherieeinheiten, welche bei den dazugehörigen DMA1 und DMA2 Kanälen (Channels) eine Anfrage (Request) stellen können.

DMA1:

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
ADC1	ADC1						
SPI/I ² S		SPI1_RX	SPI1_TX	SPI2/I2S2_RX	SPI2/I2S2_TX		
USART		USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
I ² C				I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX
TIM1		TIM1_CH1	TIM1_CH2	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
TIM2	TIM2_CH3	TIM2_UP			TIM2_CH1		TIM2_CH2 TIM2_CH4
TIM3		TIM3_CH3	TIM3_CH4 TIM3_UP			TIM3_CH1 TIM3_TRIG	
TIM4	TIM4_CH1			TIM4_CH2	TIM4_CH3		TIM4_UP

DMA2:

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5
ADC3 ⁽¹⁾					ADC3
SPI/I2S3	SPI/I2S3_RX	SPI/I2S3_TX			
UART4			UART4_RX		UART4_TX
SDIO ⁽¹⁾				SDIO	
TIM5	TIM5_CH4 TIM5_TRIG	TIM5_CH3 TIM5_UP		TIM5_CH2	TIM5_CH1
TIM6/ DAC_Channel1			TIM6_UP/ DAC_Channel1		
TIM7/ DAC_Channel2				TIM7_UP/ DAC_Channel2	
TIM8 ⁽¹⁾	TIM8_CH3 TIM8_UP	TIM8_CH4 TIM8_TRIG TIM8_COM	TIM8_CH1		TIM8_CH2

6.2 Konfiguration eines DMA Transfers

Für einen DMA Transfer müssen folgende Parameter festgelegt werden:

- **Daten Transfer Richtung:**

Zunächst muss festgelegt werden, ob der Transfer zwischen Speicher und Peripherie oder zwischen zwei Speicherbereichen erfolgt. Die geschieht durch Enabeln oder Disabeln des Bits MEM2MEM im DMA Channel Configuration Register (DMA_CCRx). Bei einem Transfer zwischen Speicher und Peripherie muss noch Quelle und Ziel des Datentransfers festgelegt werden. Dies wird ebenso im DMA_CCRx festgelegt (z.B. DMA1 Kanal4: Peripherie = Source oder DMA2 Kanal2: Peripherie = Destination)

- **Basisadressen des Datentransfers:**

Im nächsten Schritt müssen die Basisadressen (Speicherbasisadresse, Peripheriebasisadresse) des Datentransfers festgelegt werden. Die Speicheradresse wird im CMARx Register des jeweiligen DMA Kanals festgelegt und die Peripheriebasisadresse im CPARx Register.

- **Memory Size:**

Die Memory Size beschreibt die Größe eines Blocks (Peripherie Memory Size, Memory Datasize) beim Datentransfer. Diese Größe wird im DMA_CCRx Register des jeweiligen DMA Kanals festgelegt. Dabei kann zwischen 8,16,32 Bit gewählt werden. Buffergröße: Die Buffergröße beschreibt Anzahl der Blöcke die transferiert werden sollen. Diese Größe wird im CNDTRx Register des jeweiligen DMA Kanals festgelegt.

- **Memory Increment, Peripheral Increment:**

Das Memory Increment bzw Peripheral Increment kann festgelegt werden, ob nach dem Transfers eines Blocks die Memory Adresse oder Peripherie Adresse der Quell/Zieladresse um die Blockgröße vor dem nächsten Blocktransfer erhöht werden soll oder nicht. Memory Increment bzw Peripheral Increment sind speziell zum Transfer von Arrays vorteilhaft.

- **DMA Mode:** Beim STM32F103 unterscheidet man zwischen zwei DMA Modi:

Circular Mode: Wenn Daten geholt wurden und der Buffer voll ist, löst der DMA einen Interrupt aus und fängt an, am Anfang des Buffers die alten Werte mit den neuen zu überschreiben (→geschlossener Kreis).

Normal Mode: Wenn Daten geholt wurde und der Buffer voll ist, löst der DMA einen Interrupt aus. Allerdings wird ab sofort kein Datentransfer ausgeführt. Um erneut einen DMA Transfer zu starten, muss man das CNDTRx Register erneut mit einer bestimmten Buffergröße konfigurieren .

Der Modus wird im DMA_CCRx Register festgelegt.

- **Priorität:**

Die jeweilige Channel Priorität kann im DMA_CRRx festgelegt werden. Dabei kann zwischen vier Prioritäten unterschieden werden (low, medium, high, very high)

- **ENABLE:**

Den Channel aktiviert man, indem man das ENABLE Bit im DMA_CRRx Register setzt. Sobald der Channel aktiviert wurde, kann er einen DMA Request liefern, sofern die richtige Peripherie zu dem Channel verwendet wird.

Im folgenden Kapitel wird gezeigt wie mithilfe des DMA Controllers DMA1 Kanal ein kontinuierlicher Transfer zwischen ADC1 und dem Speicher etabliert werden kann.

Der ADC1 wird dabei so konfiguriert das 2 ADC Kanäle (Channel 9 und Channel 14) kontinuierlich in einer regular sequence konvertiert werden. An Kanal 9 ist das Potentiometer der LED/Schalterplatine und Kanal 14 das Potentiometer des DIL Adapter des ARM Minimalsystem angeschlossen.

Der DMA Controller kopiert die konvertierten Werte in ein Array im Speicher. Die beiden Werte werden dann am LCD Display angezeigt.

6.3 Beispielprogramm #11 –DMA Controller und ADC Demoprogramm

```
/*
 * (C) Copyright HTL - HOLLABRUNN 2009-2009 All rights reserved. AUSTRIA */
 */

/* File Name:    ADC_DMA Interrupt.c */
/* Autor:        Josef Reisinger */
/* Version:      V1.00 */
/* Date:         10/01/2016 */
/* Description: Demoprogramm ADC mit DMA Controller mit ADC Demoprogramm */
/*               Funktionalitaet (PWM Out) */
/*
 * History:      V1.00 creation
 */
#include "armv10_std.h"

/* ----- Static Variables -----*/
static __IO uint16_t ADC1ConvertedValues[2]; // Array für ADC Werte

/*----- Function Prototypes -----*/
static void DMA_Config(void);
static void ADC1_Init(void);

/* ----- Definitions Reset and Clock Control -----*/
#define CFGR_ADCPRE_Reset_Mask ((uint32_t)0xFFFF3FFF)

/* ----- Definitions for DMA Channel Configuration Register -----*/
#define CCR_CLEAR_Mask          ((uint32_t)0xFFFF800F)
#define DMA_DIR_PeripheralSRC   ((uint32_t)0x00000000)
#define DMA_PeripheralInc_Disable ((uint32_t)0x00000000)
#define DMA_MemoryInc_Enable    ((uint32_t)0x00000080)
#define DMA_PeripheralDataSize_HalfWord ((uint32_t)0x00000100)
#define DMA_MemoryDataSize_HalfWord ((uint32_t)0x00000400)
#define DMA_Mode_Circular       ((uint32_t)0x00000020)
#define DMA_Priority_High       ((uint32_t)0x00002000)
#define DMA_M2M_Disable         ((uint32_t)0x00000000)
#define DMA_IT_TC               ((uint32_t)0x00000002)

/* ----- DMA Interrupt Flag Clear Register (ICFR) Definitions -----*/
#define DMA1_FLAG_TC1           ((uint32_t)0x00000002)

/* ----- ADC Private Definitions -----*/
#define CR1_CLEAR_Mask          ((uint32_t)0xFF0FEFFF) /* CR1 register Mask */
#define CR2_ADON_Set            ((uint32_t)0x00000001) /* ADC ADON mask */
#define CR2_DMA_Set              ((uint32_t)0x00000100) /* ADC DMA mask */
#define CR2_RSTCAL_Set          ((uint32_t)0x00000008) /* ADC RSTCAL mask */
#define CR2_CAL_Set              ((uint32_t)0x00000004) /* ADC CAL mask */
#define CR2_EXTTRIG_SWSTART_Set ((uint32_t)0x00500000) /* ADC Software start */
#define CR2_CLEAR_Mask           ((uint32_t)0xFFF1F7FD) /* CR2 register Mask */
#define SQR3_SQ_Set              ((uint32_t)0x0000001F) /* ADC SQx mask */
#define SQR1_CLEAR_Mask          ((uint32_t)0xFF0FFFFF) /* SQR1 register Mask */
#define SMPR1_SMP_Set             ((uint32_t)0x00000007) /* ADC SMPx mask */
#define SMPR2_SMP_Set             ((uint32_t)0x00000007)

#define ADC1_DR_ADDRESS          ((uint32_t)0x4001244C) /* ADC1 DR reg base address */

/* ----- ADC Modes -----*/
#define ADC_Mode_Independent     ((uint32_t)0x00000000)
#define ADC_DataAlign_Right      ((uint32_t)0x00000000)
#define ADC_ExternalTrigConv_None ((uint32_t)0x000E0000) /* ADC1, ADC2, ADC3 */
#define ADC_Channel_9              ((uint8_t)0x09)
#define ADC_Channel_14              ((uint8_t)0x0E)
#define ADC_SampleTime_55Cycles5 ((uint8_t)0x05)
```

```

/*-----*/
/*      Initialisiert DMA Controller für ADC um Sensorwerte zu kopieren      */
/*-----*/
static void DMA_Config(void){
uint32_t tmpreg = 0;

/* -- Initialize the DMA1 Channel1 according to the DMA_InitStructure members */
RCC->AHBENR |= RCC_AHBENR_DMA1EN;                                /* Enable the DMA1 clock */

/*----- Initialize DMA1 Channel 1 -----*/
tmpreg = DMA1_Channel1->CCR;                                     /* Get the DMA1_Channel1 CCR value */
tmpreg &= CCR_CLEAR_Mask; /*Clear MEM2MEM,PL,MSIZE,PSIZE,MINC,PINC,CIRC,DIR bits */
tmpreg |= DMA_DIR_PeripheralSRC /* DIR Bit:copy data from Peripheral to Memory */
    DMA_Mode_Circular | /* Circular Mode, better for SCAN and CONT. ADC modes */
    DMA_PeripheralInc_Disable | /* No increment for Periph. Address Reg. (CPAR) */
    DMA_MemoryInc_Enable | /* Incr. Mem Register Address after transfer (CMAR) */
    DMA_PeripheralDataSize_HalfWord /* Size of Peripheral Data Unit (16 Bit) */
    DMA_MemoryDataSize_HalfWord | /* Size of Memory Data Unit to be copied to */
    DMA_Priority_High | /* High Priority Level */
    DMA_M2M_Disable; /* No Copy from Memory to Memory */
DMA1_Channel1->CCR = tmpreg; /* Write to DMA1 Channel1 Control Register (CCR) */

DMA1_Channel1->CNDTR=2; /* Number Data Units copied in DMA_PeripheralDataSize */
DMA1_Channel1->CPAR=ADC1_DR_ADDRESS; /* peripheral base address for DMA1 Ch1 */
DMA1_Channel1->CMAR=&ADC1ConvertedValues[0]; /* Memory base address for DMA1 Ch1 */
DMA1_Channel1->CCR |= DMA_CCR1_EN; /* Enable DMA1 Channel1 */
}

/*-----*/
/*      Initialize ADC1 for Regular Sequence      */
/*-----*/
static void ADC1_Init(void){
uint32_t tmpreg = 0;
uint32_t tmpreg1 = 0;
uint32_t tmpreg2 = 0;

/* ----- Configure Clock of ADC 1 -----*/
tmpreg = RCC->CFGR;
tmpreg &= CFGR_ADCPRE_Reset_Mask; /* Clear ADCPRE[1:0] bits */
tmpreg |= RCC_CFGR_ADCPRE_DIV6; /* Set ADC Clock Source ADCPRE[1:0] bits */
RCC->CFGR = tmpreg; /* Store the new value */

/* ----- Enables the High Speed APB (APB2) peripheral clock -----*/
RCC->APB2ENR |= RCC_APB2ENR_IOPBEN | RCC_APB2ENR_IOPCEN | RCC_APB2ENR_ADC1EN;

/* ----- Set Configuration Bits for PC4 und PB1 to Analog In Mode -----*/
GPIOB->CRL &= ~0x000000F0; /* Set Mode to Analog In for PB1 (ADC12_IN9) */
GPIOC->CRL &= ~0x000F0000; /* Set Mode to Analog In for PC4 (ADC12_IN14) */

/* ----- ADC1 CR1 Configuration -----*/
tmpreg1 = ADC1->CR1; /* Get the ADCx CR1 value */
tmpreg1 &= CR1_CLEAR_Mask; /* Clear ADC DUALMOD and SCAN bits */
/* - Config the ADC to operate in independent mode, Enable Scan Conversion Mode */
/* In Scan mode, the inputs selected through the ADC_SQRx Registers */
tmpreg1 |= ADC_Mode_Independent | (0x1<<8);
ADC1->CR1 = tmpreg1; /* Write to ADC1 CR1 */

/*----- ADC1 CR2 Configuration -----*/
tmpreg1 = ADC1->CR2; /* Get the ADCx CR2 value */
tmpreg1 &= CR2_CLEAR_Mask; /* Clear CONT, ALIGN and EXTSEL bits */
tmpreg1 |= ADC_DataAlign_Right | /* ADC1 data alignment right */
    ADC_ExternalTrigConv_None | /* No external trigger used to start ADC */
Conversion of regular channels -> SW Start Conversion */
    (0x1<<1); /* Conversion in Continuous mode and not in single mode */
ADC1->CR2 = tmpreg1; /* Write to ADCx CR2 */
}

```

```

/*----- ADCx SQR1 Configuration -----*/
tmpreg1 = ADC1->SQR1;                                /* Get the ADC1 SQR1 value */
tmpreg1 &= SQR1_CLEAR_Mask;                            /* Clear L bits */
tmpreg2 |= (uint8_t)(0x2-0x1);      /* Regular channel sequence with 2 channels */
tmpreg1 |= (uint32_t)tmpreg2 << 20;
ADC1->SQR1 = tmpreg1;                                /* Write to ADC1 SQR1 */

/* ----- Define Channel 9 as 1st item for ADC 1 regular sequence -----*/
tmpreg1 = ADC1->SMPR2;                                /* Get the old register value */
tmpreg2 = SMPR2_SMP_Set <<(3*ADC_Channel_9);        /* Calculate the mask to clear */
tmpreg1 &= ~tmpreg2;                                 /* Clear the old channel sample time */
tmpreg2 = ADC_SampleTime_55Cycles5 << (3*ADC_Channel_9);
tmpreg1 |= tmpreg2;                                  /* Set Ch9 and sample time */
ADC1->SMPR2 = tmpreg1;                            /* Store the new register value */

tmpreg1 = ADC1->SQR3; /* Get the old register value */
tmpreg2 = SQR3_SQ_Set << (5*(1-1));    /* mask to clear for Channel 9 in rank 1*/
tmpreg1 &= ~tmpreg2;                         /* Clear the old SQR bits for the selected rank */
tmpreg2 = ADC_Channel_9 << (5*(1-1));    /* mask to set Ch9 in rank 1*/
tmpreg1 |= tmpreg2;                          /* Set the SQRx bits for the selected rank */
ADC1->SQR3 = tmpreg1;                      /* Store the new register value */

/* ----- Define Channel 14 as 2nd item for ADC 1 regular sequence -----*/
tmpreg1 = ADC1->SMPR1;                                /* Get the old register value */
tmpreg2 = SMPR1_SMP_Set << (3*(ADC_Channel_14-10)); /* mask to clear */
tmpreg1 &= ~tmpreg2;                                 /* Clear the old channel sample time */
tmpreg2 = ADC_SampleTime_55Cycles5 << (3*(ADC_Channel_14-10)); /* mask to set */
tmpreg1 |= tmpreg2;                                  /* Set Channel 14 and sample time */
ADC1->SMPR1 = tmpreg1;                            /* Store the new register value */

tmpreg1 = ADC1->SQR3;                                /* Get the old register value */
tmpreg2 = SQR3_SQ_Set << (5*(2-1));    /* mask to clear for Ch14 in rank 2 */
tmpreg1 &= ~tmpreg2;                         /* Clear the old SQR bits for the selected rank */
tmpreg2 = ADC_Channel_14 << (5*(2-1));    /* mask to set for Channel 14 in rank 2*/
tmpreg1 |= tmpreg2;                          /* Set the SQR3 bits for the selected rank */
ADC1->SQR3 = tmpreg1;                      /* Store the new register value */

ADC1->CR2 |= CR2_DMA_Set;          /* Enable the selected ADC1 DMA request */
ADC1->CR2 |= CR2_ADON_Set;        /* ADON =1-> wake up the ADC1 from power down mode */

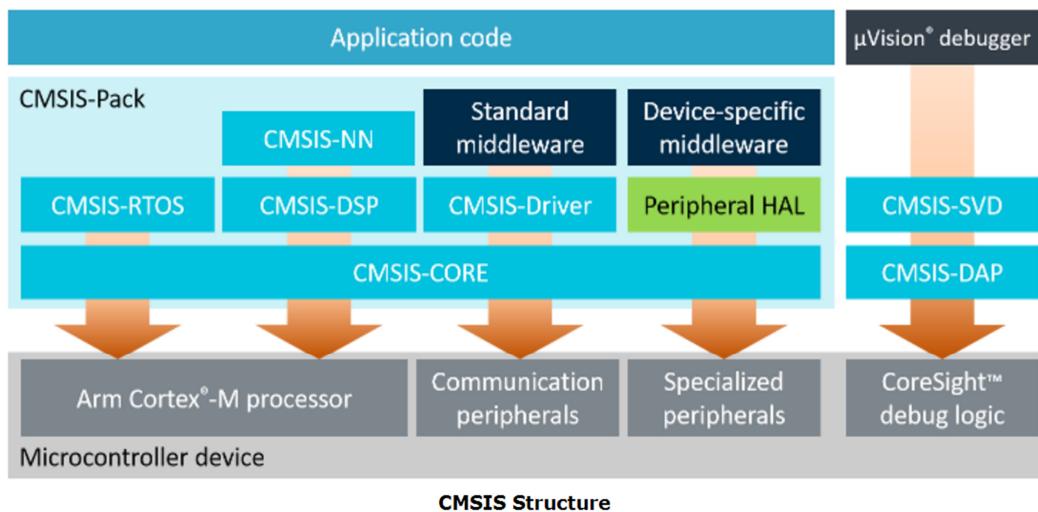
/* ----- Calibration of ADC 1 -----*/
ADC1->CR2 |= CR2_RSTCAL_Set;        /* Clear ADC1 calibration registers */
while((ADC1->CR2 & CR2_RSTCAL_Set) != 0); /* Calibration registers initialized ? */
ADC1->CR2 |= CR2_CAL_Set;           /* Start Calibration of ADC 1 */
while((ADC1->CR2 & CR2_CAL_Set) != 0); /* Calibration finished ? */

ADC1->CR2 |= CR2_EXTTRIG_SWSTART_Set; /*SW Start Conversion of regular channel */
}

```

```
*****  
/* MAIN function */  
*****  
int main (void)  
{  
    __IO uint16_t Ch_9, Ch_14 = 0;  
    char buffer[100],x;  
  
    lcd_init(); lcd_clear();  
    init_leds_switches(); /* Leds und Schalter initialisieren */  
    set_leds(0); /*Leds auf 0 setzen */  
    DMA_Config(); /* Init DMA1 Channel1 to copy ADC1 Data of Ch9 And Ch14 to MEM */  
    ADC1_Init(); /* Init ADC1: regular sequence (Ch9,Ch14) in Scan Mode*/  
    do {  
        set_leds(x); /* Running LEDS every 500 ms */  
        x++;  
        wait_ms(500);  
        if ((Ch_9 != ADC1ConvertedValues[0]) || (Ch_14 != ADC1ConvertedValues[1])) {  
            Ch_9 = ADC1ConvertedValues[0];  
            Ch_14 = ADC1ConvertedValues[1];  
            lcd_set_cursor(0,0);  
            sprintf (&buffer[0], "Ch09=%3x",Ch_9);  
            lcd_put_string(&buffer[0]); /* Display ADC1 Ch9 Value on LCD in row 1*/  
            lcd_set_cursor(1,0);  
            sprintf (&buffer[0], "Ch14=%3x",Ch_14);  
            lcd_put_string(&buffer[0]);/* Display ADC1 Ch14 Value on LCD in row 1  
        }  
    } while (1==1);  
}
```

7 CMSIS-Cortex Mikrocontroller Software Interface Standard



CMSIS (ARM® Cortex™ Microcontroller Software Interface Standard):

Im November 2008 entwickelte ARM gemeinsam mit IAR Systems, Segger, Micrium, Atmel, Luminary Micro (jetzt TI), NXP Semiconductors und STMicroelectronics den so genannten CMSIS – Cortex Microcontroller Software Interface Standard. Dabei handelt es sich um einen einheitlichen, herstellerunabhängigen Abstraktions Layer für alle Cortex Prozessoren der Firma ARM der auch den Zugriff auf die Peripherie regelt, welche ja ein wichtiges Differenzierungsmerkmal zwischen den verschiedenen Halbleiterherstellern darstellt. Die aktuelle Version ist die Version 5.0. CMSIS besteht aus folgenden Komponenten: <https://www.keil.com/pack/doc/CMSIS/General/html/index.html>

CMSIS Core (Cortex-M):

Der CMSIS Core stellt ein Interface (API) für den Prozessore Core (Cortex-M0, Cortex-M0+, Cortex-M3, Cortex-M4, Cortex-M7, Cortex-M23, Cortex-M33, Cortex-M35P, SC000, and SC300) und der Peripherieeinheiten sowie Interrupts bereit. Im Rahmen des CMSIS-Standards wurden die Headerdateien standardisiert, der Zugriff auf die Register erfolgt per **Peripheral->Register**. Die CMSIS C-Dateien bzw. Header enthalten auch Anpassungen für die verschiedenen Compiler. Der CMSIS Core enthält auch SIMD intrinsic Funktionen für Cortex-M4, Cortex-M7, Cortex-M33 und Cortex-M35P.

CMSIS Driver:

Definieren ein generisches Treiber Interface für die Middleware und machen dieses nutzbar für alle unterstützten Devices. Dieses API ist RTOS unabhängig und stellt eine Verbindung der Microcontroller Peripherie zur Middleware her (z.B: CAN, Ethernet, USB, I2C, SPI, USART, Flash, FileSystem..)

CMSIS NN:

CMSIS-NN ist ein Sammlung für Neural Network Kernels, die optimiert sind für maximale Performance und einen kleinen Speicher Footprint für die Cortex-M Prozessor Cores.

CMSIS DSP:

Die DSP Library stellt über 60 Funktionen zur Signalverarbeitung für verschiedene Datentypen wie fixpoint (fractional q7, q15, q31) und single precision floating-point (32-bit) bereit. Die Library ist für alle Cortex-M3 verfügbar und ist optimiert für alle SIMD Befehle, die für Cortex-M4, Cortex-M7, Cortex-M33 und Cortex-M35P.

CMSIS RTOS API:

Stellt ein API Standard für Realtime Operating Systems, der portierbar ist für viele RTOS Kernels. CMSIS RTOS API wird momentan vom RTX-Kernel von Keil unterstützt.

CMSIS SVD:

System View Description XML Files enthalten die Beschreibungen des kompletten Mikrocontrollersystems einschließlich der Peripherieeinheiten aus Programmierersicht. Diese werden im Debugger genutzt um die entsprechende Registerinformation zu veranschaulichen.

CMSIS DAP:

Debug Access Port. Standardisierte Firmware für eine Debug Einheit, die den CoreSight Debug Logic mit dem DAP Interface verbindet. CMSIS DAP wird als separates package bereit gestellt, die in Evaluation Boards integriert werden kann.

8 StdPeriph Drivers für STM32F10X Microcontroller

Im Folgenden einige Beispielprogramme die zeigen wie die Peripherieeinheiten des STM32F10X unter Verwendung der STM32F10X Standard Peripheral Library (**StdPeriph**) von ST-Microelectronics benutzt werden können.

8.1 Beispielprogramm #6 – Interrupt UART

```
*****  
/* (C) Copyright HTL - HOLLABRUNN 2009-2019 All rights reserved. AUSTRIA */  
/*  
/* File Name: uart_echo_int.c */  
/* Autor: Josef Reisinger */  
/* Version: V1.00 */  
/* Date: 01/05/2019 */  
/* Description: Demoprogramm für Uart Echo using StdPeriph Device Drivers */  
*****  
/* History: V1.00 creation */  
*****  
  
#include "stm32f10x.h" // Device header  
#include "stm32f10x_rcc.h" // Keil::Device:StdPeriph Drivers:RCC  
#include "stm32f10x_gpio.h" // Keil::Device:StdPeriph Drivers:GPIO  
#include "stm32f10x_usart.h" // Keil::Device:StdPeriph Drivers:USART  
  
*****  
/* UART1 ISR Handler: echo received characters */  
*****  
void USART1_IRQHandler()  
{  
    // echo received data from USART1  
    USART_SendData(USART1, USART_ReceiveData(USART1));  
}  
  
*****  
/* Send String via USARTx */  
*****  
void USART_SendString(USART_TypeDef *USARTx, char *str)  
{  
    // Sends a string, character for character, over the specified USART  
    while (*str) {  
        while (USART_GetFlagStatus(USARTx, USART_FLAG_TXE) == RESET);  
        USART_SendData(USARTx, *str++);  
    }  
}  
  
*****  
/* Main Programm */  
*****  
int main()  
{  
    GPIO_InitTypeDef gpio;  
    USART_ClockInitTypeDef usartclock;  
    USART_InitTypeDef usart;  
    NVIC_InitTypeDef nvic;  
  
    // Enable all GPIO and USART clocks needed for USART1  
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);  
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);  
  
    // Create gpio struct and fill it with defaults
```

```
GPIO_StructInit(&gpio);

// Set PA9 to alternate function push pull (Tx)
gpio.GPIO_Mode = GPIO_Mode_AF_PP;
gpio.GPIO_Pin = GPIO_Pin_9;
GPIO_Init(GPIOA, &gpio);

// Set PA10 to input floating (Rx)
gpio.GPIO_Mode = GPIO_Mode_IN_FLOATING;
gpio.GPIO_Pin = GPIO_Pin_10;
GPIO_Init(GPIOA, &gpio);

// Create usart struct and init USART1 to 115 200 baud
USART_StructInit(&usart);
usart.USART_BaudRate = 115200;
usart.USART_WordLength = USART_WordLength_8b;
usart.USART_StopBits = USART_StopBits_1;
usart.USART_Parity = USART_Parity_No;
usart.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
usart.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
USART_Init(USART1, &usart);

// Init USART1 clock
USART_ClockStructInit(&usartclock);
USART_ClockInit(USART1, &usartclock);

// Init NVIC for USART1 RXNE
nvic.NVIC_IRQChannel = USART1_IRQn;
nvic.NVIC_IRQChannelCmd = ENABLE;
nvic.NVIC_IRQChannelPreemptionPriority = 0;
nvic.NVIC_IRQChannelSubPriority = 2;
NVIC_Init(&nvic);
USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);

// Enable USART1
USART_Cmd(USART1, ENABLE);

USART_SendString(USART1, "Welcome USART1 Echo V1.0\xd\xa");

for (;;)
{
    // do nothing...
}

}
```

8.2 Beispielprogramm #7 – Externer Interrupt

```
/*********************************************
/* (C) Copyright HTL - HOLLABRUNN 2009-2009 All rights reserved. AUSTRIA */
/*
/* File Name: External_Interrupt.c
/* Autor: Josef Reisinger
/* Version: V1.00
/* Date: 11/03/2012
/* Description: Demoprogramm für External Interrupt
/*********************************************
/* History: V1.00 creation
/********************************************/

#include "stm32f10x.h" // Device header
#include <armv10_std.h>
#include "stm32f10x_rcc.h" // Keil::Device:StdPeriph Drivers:RCC
#include "stm32f10x_gpio.h" // Keil::Device:StdPeriph Drivers:GPIO
#include "stm32f10x_exti.h" // Keil::Device:StdPeriph Drivers:EXTI

/*----- Function Prototypes -----*/
static void EXTI1_Config(void);

/*----- Static Variables-----*/
int falling_edges;

/*********************************************
/* External Interrupt Service Routine Schalter1 */
/*********************************************
void EXTI1_IRQHandler(void) //ISR
{
    EXTI_ClearITPendingBit(EXTI_Line1);
    //Pending bit EXTI1 rücksetzen (Sonst wird die ISR immer wiederholt)
    falling_edges++; // Fallende Flanke an PA1 erkannt
    return;
}

/*********************************************
/* EXTI1_config
/* Leitung PA1 wird mit EXTI1 verbunden, Interupt bei falling edge, Priorität 3 */
/*********************************************
static void EXTI1_Config(void) {
    GPIO_InitTypeDef gpio;
    EXTI_InitTypeDef EXTI_InitStruct;
    NVIC_InitTypeDef nvic;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE); // GPIOA Clock Enable

    GPIO_StructInit(&gpio); // Create gpio struct and fill it with defaults
    gpio.GPIO_Mode = GPIO_Mode_IPU; // Configure PA1 to input Pull UP Mode
    gpio.GPIO_Pin = GPIO_Pin_1;
    GPIO_Init(GPIOA, &gpio);

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
    //AFIOEN - Clock enable
    EXTI_DeInit();

    EXTI_StructInit(&EXTI_InitStruct);
    EXTI_InitStruct.EXTI_Line = EXTI_Line1;
    EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Falling;
    EXTI_InitStruct.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStruct); /* save initialisation */
}
```

```

GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource1);
/* configure EXTI1 Port A */

EXTI_ClearITPendingBit(EXTI_Line1);
//EXTI_clear_pending: Das Auslösen auf vergangene Vorgänge nach dem enablen
//verhindern

// Init NVIC for EXTI1 Interrupt
nvic.NVIC_IRQChannel = EXTI1 IRQn;
nvic.NVIC_IRQChannelCmd = ENABLE;
nvic.NVIC_IRQChannelPreemptionPriority = 0;
nvic.NVIC_IRQChannelSubPriority = 3;
NVIC_Init(&nvic);

}

/*********************************************
/*          MAIN function
/*****************************************/
int main (void)
{
int lcd_falling_edges;
char buffer[30];

falling_edges=0;      // Zaehler für falling edges initialisieren
lcd_falling_edges=0;

lcd_init ();           // Initialisieren der LCD Anzeige
lcd_clear();           // LCD Anzeige löschen
sprintf(&buffer[0],"Fall. Edges=%d", lcd_falling_edges);
// zaehler auf LCD aktualisieren
lcd_put_string(&buffer[0]);

EXTI1_Config();
// Konfigurieren des Externalen Interrupts für Leitung PA1 (Falling Edges)
do
{
    if (lcd_falling_edges != falling_edges)    // Anzeigewert geändert?
    {
        lcd_falling_edges = falling_edges;
        lcd_set_cursor(0,0);      // Cursor auf Ursprung
        sprintf(&buffer[0],"Fall. Edges=%d", lcd_falling_edges);
        // zaehler auf LCD aktualisieren
        lcd_put_string(&buffer[0]);
    }
} while (1);

}

```

8.3 Beispielprogramm #8 – Lauflicht mit Timer Interrupt

```
*****  
/* (C) Copyright HTL - HOLLABRUNN 2009-2009 All rights reserved. AUSTRIA */  
/*  
/* File Name: Lauflicht_Interrupt.c  
/* Autor: Josef Reisinger  
/* Version: V1.00  
/* Date: 11/03/2012  
/* Description: Demoprogramm für Timer 1  
*****  
/* History: V1.00 creation */  
*****  
#include <stm32f10x.h>  
#include <armv10_std.h>  
#include "stm32f10x_gpio.h"  
#include "stm32f10x_rcc.h"  
#include "stm32f10x_tim.h"  
/*----- Function Prototypes -----*/  
static void TIM1_Config(void);  
  
/*----- Static Variables-----*/  
static int sekunden;  
  
*****  
/* Interrupt Service Routine Timer1 (General Purpose Timer) */  
*****  
void TIM1_UP_IRQHandler(void) //Timer 1, löst alle 1000ms aus  
{  
    TIM_ClearFlag(TIM1, TIM_FLAG_Update);  
    // Clear Update Interrupt Flag (UIF) Flag  
    sekunden++;  
}  
  
*****  
/* Configure Timer 1 */  
*****  
static void TIM1_Config(void)  
{  
    TIM_TimeBaseInitTypeDef TIM_TimeBase_InitStructure;  
    NVIC_InitTypeDef nvic;  
  
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);  
  
    TIM_DeInit(TIM1);  
    TIM_TimeBase_InitStructure.TIM_ClockDivision = TIM_CKD_DIV1;  
    TIM_TimeBase_InitStructure.TIM_CounterMode = TIM_CounterMode_Up;  
    /* T_INT = 13,8ns, Annahme: Presc = 1350 ---> Auto Reload Wert = 52801  
    (=0xCE41) ---> 1s Update Event*/  
    //Auto-Reload Wert = Maximaler Zaehlerstand des Upcounters  
    TIM_TimeBase_InitStructure.TIM_Period = 0xCE41;  
    //Wert des prescalers (Taktverminderung)  
    TIM_TimeBase_InitStructure.TIM_Prescaler = 1350;  
    //Repetition Counter deaktivieren  
    TIM_TimeBase_InitStructure.TIM_RepetitionCounter = 0;  
    TIM_TimeBaseInit(TIM1, &TIM_TimeBase_InitStructure);  
  
    TIM_ITConfig (TIM1, TIM_DIER_UIE,ENABLE); // Update Interrupt Enable  
  
    // Init NVIC for Timer 1 Update Interrupt  
    nvic.NVIC_IRQChannel = TIM1_UP_IRQn;  
    nvic.NVIC_IRQChannelCmd = ENABLE;  
    nvic.NVIC_IRQChannelPreemptionPriority = 0;  
    nvic.NVIC_IRQChannelSubPriority = 2;
```

```
NVIC_Init(&nvic);

    TIM_Cmd(TIM1, ENABLE); //Counter-Enable bit (CEN) setzen
}

/*********************************************
 *          MAIN function
 ****/
/********************************************/

int main (void)
{
int lauflicht;           // Lauflicht P1 Abbild
int lcd_sekunden;        // aktueller Zählerstand auf LCD
char buffer[30];

init_leds_switches();
lcd_init();              // LCD initialisieren
lcd_clear();             // Lösche LCD Anzeige
lauflicht = 0x01; // Lauflicht im Speicher initialisieren
sekunden=0;              // Sekundenzaehler initialisieren
lcd_sekunden = 0; // aktueller Anzeigewert
set_leds(0);             // LED löschen
// Timer 1 starten: Upcounter --> löst alle 1s einen Update Interrupt
TIM1_Config();
do
{
    if (lcd_sekunden != sekunden) // 1 Sekunde vergangen ?
    {
        lcd_sekunden = sekunden;
        set_leds(lauflicht); // Lauflicht auf LED aktualisieren
        lauflicht = lauflicht*2; // naechtes Lauflicht Bit
        if (lauflicht==256) // letztes Lauflicht Bit ?
        {
            lauflicht = 0x01; // Anfangszustand Lauflicht
        }
        lcd_set_cursor(0,0); // Cursor auf Ursprung
        // Sekundenzaehler auf LCD aktualisieren
        sprintf(&buffer[0],"Sekunden=%d", lcd_sekunden);
        lcd_put_string(&buffer[0]);
    }
} while (1);
}
```

8.4 Beispielprogramm #9 –Input Capture Demoprogramm

```
/*
 * (C) Copyright HTL - HOLLABRUNN 2009-2019 All rights reserved. AUSTRIA */
/* File Name: Timer_Input_Capture.c */
/* Autor: Josef Reisinger */
/* Version: V1.00 */
/* Date: 13/05/2019 */
/* Description: Demoprogramm für Capture Eingang an GP Timer 2 mit STDPeriph */
/* Library */
/* History: V1.00 creation */
#include "armv10_std.h"

/*----- Function Prototypes -----*/
static void TIM2_Config(void);

/*----- Static Variables-----*/
__IO uint16_t IC2ReadValue1 = 0, IC2ReadValue2 = 0;
__IO uint16_t CaptureNumber = 0;
__IO uint32_t Capture = 0;

/*
 * Timer 2 Interrupt Service Routine
 */
void TIM2_IRQHandler(void)
{
    if ((TIM_GetFlagStatus(TIM2, TIM_FLAG_Update) != RESET) &&
        (TIM_GetITStatus(TIM2, TIM_IT_Update) != 0)) // Update Interrupt Pending ?
    {
        TIM_ClearFlag(TIM2, TIM_FLAG_Update); /* Clr TIM2 Update Int. pending bit */
        LED7=~LED7;
    }
    if ((TIM_GetFlagStatus(TIM2, TIM_FLAG_CC2) != RESET) &&
        (TIM_GetITStatus(TIM2, TIM_IT_CC2) != 0)) // Capture Interrupt von Ch2?
    {
        TIM_ClearFlag(TIM2, TIM_FLAG_CC2); /* Clr TIM2 Capture Int. pending bit Ch2
        if(CaptureNumber == 0)
        {
            IC2ReadValue1 = TIM_GetCapture2(TIM2); /* Get Input Capture value for Ch2*/
            CaptureNumber = 1;
        }
        else if(CaptureNumber == 1)
        {
            IC2ReadValue2 = TIM_GetCapture2(TIM2); /* Get Input Capture value for Ch2*/
            if (IC2ReadValue2 > IC2ReadValue1) /* Capture computation */
            {
                Capture = (IC2ReadValue2 - IC2ReadValue1);
            }
            else
            {
                Capture = ((TIM2->ARR - IC2ReadValue1) + IC2ReadValue2);
            }
            CaptureNumber = 0;
        }
    }
}

/*
 * Configure Timer 4
 */

```

```

static void TIM2_Config(void) {
GPIO_InitTypeDef gpio;
TIM_TimeBaseInitTypeDef TIM_TimeBase_InitStructure;
TIM_ICInitTypeDef TIM_ICInitStructure;
NVIC_InitTypeDef nvic;

RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE); // GPIOA Clock Enable
GPIO_StructInit(&gpio); // Create gpio struct and fill it with defaults
gpio.GPIO_Mode = GPIO_Mode_IPU; // PA1(=SW1) bzw. CH2 Timer 2:Mode Input Pull UP
gpio.GPIO_Pin = GPIO_Pin_1;
GPIO_Init(GPIOA, &gpio);

RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE); // Clock Enable Timer 2
/* ----- Configure Timer 2-----*/
TIM_DeInit(TIM2);
TIM_TimeBase_InitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
TIM_TimeBase_InitStructure.TIM_CounterMode = TIM_CounterMode_Up;
/* - T_INT = 13,8ns, Annahme: Presc = 35712 --> Auto Reload Wert = 2000 --> 1s*/
TIM_TimeBase_InitStructure.TIM_Period = 2000; // Max. Zaehlerstand UpCounter
TIM_TimeBase_InitStructure.TIM_Prescaler = 35712; // system clock to 2kHz
TIM_TimeBaseInit(TIM4, &TIM_TimeBase_InitStructure);

/* ----- Configure TIM2 CH2 (PA1) as input capture channel-----*/
TIM_ICInitStructure.TIM_Channel = TIM_Channel_2;
TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;
TIM_ICInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI;
TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;
TIM_ICInitStructure.TIM_ICFilter = 0xF; /* filter to avoid bouncing */
TIM_ICInit(TIM2, &TIM_ICInitStructure);

TIM_ITConfig (TIM2, TIM_IT_Update,ENABLE); // Timer 2 Update Interrupt Enable
TIM_ITConfig (TIM2, TIM_IT_CC2,ENABLE); // Enable TIM2 CC2 Interrupt Ch2

// Init NVIC for Timer 2 Interrupt
nvic.NVIC_IRQChannel = TIM2_IRQn;
nvic.NVIC_IRQChannelCmd = ENABLE;
nvic.NVIC_IRQChannelPreemptionPriority = 0;
nvic.NVIC_IRQChannelSubPriority = 2;
NVIC_Init(&nvic);

TIM_Cmd(TIM2, ENABLE); //Counter-Enable bit (CEN) Timer 2 setzen
}
/*********************************************
*                                     MAIN function
*****************************************/
int main (void) {
char buffer[50];

lcd_init(); // LCD Anzeige initialisieren
TIM2_Config(); // Configure Timer 2

do {

wait_ms(900);
lcd_clear();
lcd_set_cursor(0,0);
sprintf (&buffer[0], "CNT=%d", TIM2->CNT);
lcd_put_string(&buffer[0]);
lcd_set_cursor(1,0);
sprintf (&buffer[0], "%d:%d ", IC2ReadValue1, IC2ReadValue2);
lcd_put_string(&buffer[0]);

} while (1==1);
}

```

8.5 Beispielprogramm #10 –Output Compare mit PWM Demoprogramm

```
/*********************************************
/* (C) Copyright HTL - HOLLABRUNN 2009-2009 All rights reserved. AUSTRIA */
/*
/* File Name: Timer_Compare_PWM.c
/* Autor: Josef Reisinger
/* Version: V1.00
/* Date: 08/05/2019
/* Description: LED Dimmer: Demoprogramm für Timer Compare
/* Funktionalitaet (PWM Out) unter Nutzung von StdPeriph
/*********************************************
/* History: V1.00 creation
/*********************************************
#include <stm32f10x.h>
#include "stm32f10x_rcc.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_tim.h"

/*----- Function Prototypes -----
static void TIM4_Config(void);

/*----- Static Variables-----
/*********************************************
/* Timer 4 Interrupt Service Routine
/*********************************************
void TIM4_IRQHandler(void) { //Timer 4 = PWM Output Mode, Channel 3
    static char direction=1; //Auf- / Abdimmern

    TIM4->SR &=~0x00FF; //Interrupt pending-bit löschen
    if(direction==0) {
        /* Increment Compare Value Channel 3 */
        TIM_SetCompare3(TIM4, TIM_GetCapture3(TIM4)+1);
        if(TIM_GetCapture3(TIM4)>=(TIM4->ARR/2)) {
            direction=1; //Danach abdimmen
        }
    }
    else {
        /* Decrement Compare Value Channel 3 */
        TIM_SetCompare3(TIM4, TIM_GetCapture3(TIM4)-1);
        if(TIM_GetCapture3(TIM4)==0) {
            direction=0; //Danach aufdimmen
        }
    }
    return;
}

/*----- Configure Timer 4 -----
static void TIM4_Config(void)
{
    GPIO_InitTypeDef gpio;
    TIM_TimeBaseInitTypeDef TIM_TimeBase_InitStructure;
    TIM_OCIInitTypeDef TIM_OCIInitStruct;
    NVIC_InitTypeDef nvic;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE); // GPIOB Clock Enable

    GPIO_StructInit(&gpio); // Create gpio struct and fill it with defaults
    gpio.GPIO_Mode = GPIO_Mode_AF_PP; // PB8(=LED0, TIM4_CH3) in AF -Push Pull
    gpio.GPIO_Pin = GPIO_Pin_8;
    GPIO_Init(GPIOB, &gpio);
}
```

```

RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE); // Clock Enable Timer 4

TIM_DeInit(TIM4);
TIM_TimeBase_InitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
TIM_TimeBase_InitStructure.TIM_CounterMode = TIM_CounterMode_Up;
/* ----- T_INT = 13,8ns, Annahme: Presc = 0x925 --> Auto Reload Wert =
0x00FF (=256) --> 8,27ms*/
TIM_TimeBase_InitStructure.TIM_Period = 0xFF; //Auto-Reload Wert =
Maximaler Zaehlerstand des Upcounters
TIM_TimeBase_InitStructure.TIM_Prescaler = 0x925; //Wert des prescalers
(Taktverminderung)
TIM_TimeBaseInit(TIM4, &TIM_TimeBase_InitStructure);

/* initialize Output Compare Channels */
TIM_OCInitStruct.TIM_OCMode = TIM_OCMode_PWM1;
TIM_OCInitStruct.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStruct.TIM_Pulse = (TIM_TimeBase_InitStructure.TIM_Period)/2;
/* Compare Value*/
TIM_OCInitStruct.TIM_OCPolarity = TIM_OCPolarity_High;
TIM_OC3Init(TIM4,&TIM_OCInitStruct); /* save initialisation */

TIM_ITConfig (TIM4, TIM_IT_Update,ENABLE); // Timer 4 Update Interrupt
Enable

// Init NVIC for Timer 1 Update Interrupt
nvic.NVIC_IRQChannel = TIM4_IRQn;
nvic.NVIC_IRQChannelCmd = ENABLE;
nvic.NVIC_IRQChannelPreemptionPriority = 0;
nvic.NVIC_IRQChannelSubPriority = 2;
NVIC_Init(&nvic);

TIM_Cmd(TIM4, ENABLE); //Counter-Enable bit (CEN) Timer 4 setzen
}

/*********************************************
*          MAIN function
*****
int main (void)
{
    TIM4_Config(); // Timer 4 konfigurieren (PWM Modus auf Kanal 3)
    /*----- Timer 4 Starten -----
*/
do {
    } while (1==1);
}

```

8.6 Beispielprogramm #11 –DMA Controller und ADC Demoprogramm

```
/*
 * (C) Copyright HTL - HOLLABRUNN 2009-2009 All rights reserved. AUSTRIA */
/*
/* File Name: ADC_DMA_Interrupt.c */
/* Autor: Josef Reisinger */
/* Version: V1.00 */
/* Date: 10/01/2016 */
/* Description: Demoprogramm ADC mit DMA Controller mit ADC Demoprogramm */
/* Funktionalitaet unter Nutzung von STDPeriph Drivers */
/*
/* History: V1.00 creation */
/*
#include "stm32f10x.h"
#include "armv10_std.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_dma.h"

/*----- Definitions -----*/
#define ADC1_DR_ADDRESS ((uint32_t)0x4001244C)

/*----- Static Variables -----*/
static __IO uint16_t ADC1ConvertedValues[2]; // Array für ADC Werte

/*----- Function Prototypes -----*/
static void DMA_Config(void);
static void ADC1_Init(void);

/*
 * Function Name : DMA1_Channel1_IRQHandler
 * Description   : This function handles DMA1 Channel 1 interrupt request.
 * Input         : None
 * Output        : None
 * Return        : None
*/
void DMA1_Channel1_IRQHandler(void)
{
    /* Clears the corresponding TCIF flag (Transmission Complete) */
    ClearFlag(DMA1_FLAG_TC1);
}

/*
 *      Initialisiert DMA Controller für ADC um Sensorwerte zu kopieren
*/
static void DMA_Config(void){

DMA_InitTypeDef DMA_InitStructure;
NVIC_InitTypeDef nvic;

/* Initialize the DMA1 Channel1 according to the DMA_InitStructure members */
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE); // Enable the DMA1 clock

DMA_DeInit(DMA1_Channel1); // Deinitialize DMA Channel 1

DMA_InitStructure.DMA_PeripheralBaseAddr = ADC1_DR_ADDRESS; //Source Address
// REJ: Destination fpr copy process
DMA_InitStructure.DMA_MemoryBaseAddr =(uint32_t)&ADC1ConvertedValues;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
DMA_InitStructure.DMA_BufferSize = 2; // copy two 16bit ADC Values to Memory
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
// Holt 16Bit Wert
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
}
```

```

//Better for SCAN and CONTINOUS ADC modes
DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
DMA_Init(DMA1_Channel1, &DMA_InitStructure);

DMA1_Channel1->CCR |= DMA1_IT_TC1; //Transmition-complete --> Interrupt
DMA_ClearITPendingBit(DMA1_FLAG_TC1); //Interrupt-Bit löschen
DMA_Cmd(DMA1_Channel1, ENABLE); /* Enable DMA1 Channel1 */

// Init NVIC for DMA1 Transmition-complete --> Interrupt (Serves ADC1)
nvic.NVIC_IRQChannel = DMA1_Channel1 IRQn;
nvic.NVIC_IRQChannelCmd = ENABLE;
nvic.NVIC_IRQChannelPreemptionPriority = 0;
nvic.NVIC_IRQChannelSubPriority = 2;
NVIC_Init(&nvic);
}

/*
 *           Initialisiert ADC Channels 9 and CH14
 */
static void ADC1_Init(void){

    GPIO_InitTypeDef GPIO_InitStructure;
    ADC_InitTypeDef ADC_InitStructure;

    RCC_ADCCLKConfig(RCC_PCLK2_Div6); // Vorteiler für ADC-Takt
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_GPIOC, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

    // Pin für ADC Eingang: PB1=ADC12_IN9
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN; // PB1 -> Mode Analog In
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    // Pins für ADC Eingang: PC4=ADC12_IN14
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN; // PC4 -> Mode Analog In
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    // ADC1 für Potis konfigurieren (2 analoge Inputs)
    ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
    ADC_InitStructure.ADC_NbrOfChannel = 2; // 2 Channels: IN9,IN14
    ADC_InitStructure.ADC_ScanConvMode = ENABLE; //Enable Scan Mode
    ADC_Init(ADC1, &ADC_InitStructure);

    //Define Regual Channel Sequence (Channel 9 (PB1), Channel 14 (PC4)
    ADC-RegularChannelConfig(ADC1, ADC_Channel_9, 1, ADC_SampleTime_55Cycles5);
    ADC-RegularChannelConfig(ADC1, ADC_Channel_14, 2, ADC_SampleTime_55Cycles5);

    ADC_DMACmd(ADC1, ENABLE); /* Enable ADC1 DMA transfer */
    ADC_Cmd(ADC1, ENABLE); // ADC aktivieren

    ADC_ResetCalibration(ADC1); // Kalibrierungsvorgang der ADC1
    while(ADC_GetResetCalibrationStatus(ADC1));
    ADC_StartCalibration(ADC1);
    while(ADC_GetCalibrationStatus(ADC1));

    ADC_SoftwareStartConvCmd(ADC1, ENABLE); // trigger first conversion
}

```

```
*****  
/* MAIN function */  
*****  
int main (void)  
{  
    __IO uint16_t Ch_9 = 0;  
    __IO uint16_t Ch_14 = 0;  
    char buffer[100];  
    char x=0;  
  
    lcd_init();  
    lcd_clear();  
    init_leds_switches(); //Leds und Schalter initialisieren  
    set_leds(0); //Leds auf 0 setzen  
    DMA_Config();  
    // Initalite DMA1 Channel to transfer ADC1 Data for  
    // Channel 9 and 14 to Memory  
    ADC1_Init(); // Initalite ADC1 to convert Channel 9 and 14 continously  
    // as a regular sequence in Scan Mode  
    do {  
        set_leds(x); // Running LEDS every 400 ms  
        x++;  
        wait_ms(500);  
        if ((Ch_9 != ADC1ConvertedValues[0]) || (Ch_14 != ADC1ConvertedValues[1]))  
        // Values Changed ?  
        {  
            Ch_9 = ADC1ConvertedValues[0];  
            Ch_14 = ADC1ConvertedValues[1];  
            lcd_set_cursor(0,0);  
            sprintf (&buffer[0], "Ch09=%3x", Ch_9);  
            lcd_put_string(&buffer[0]); // Display ADC1 Channel 9 Value on LCD in row 1  
            lcd_set_cursor(1,0);  
            sprintf (&buffer[0], "Ch14=%3x", Ch_14);  
            lcd_put_string(&buffer[0]); // Display ADC1 Channel 14 Value on LCD in row 1  
        }  
    } while (1==1);  
}
```