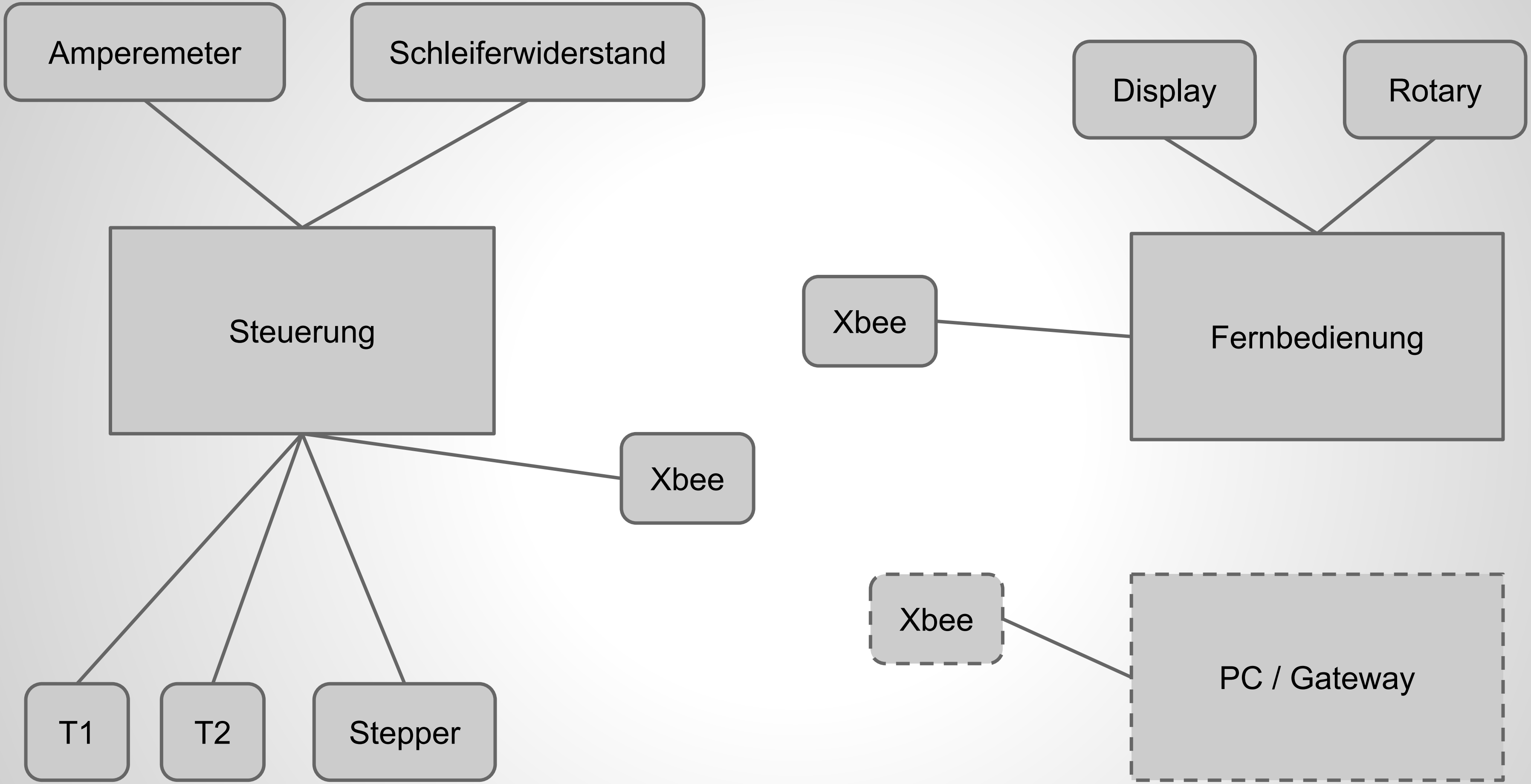
ARM Programmierung am STM32F4-Discovery

Gewächshaussteuerung
von
René Galow und Dennis Rump


Amperemeter
Schleiferwiderstand
Steuerung
Xbee
T1
T2
Stepper
Display
Rotary
Xbee
Fernbedienung
Xbee
PC / Gateway

Verwendete Technologien

- UART
- ADC
- DMA
- Interrupts
- GPIO
- 1Wire
- XBEE
- Stepper
- Rotary

DMA und UART

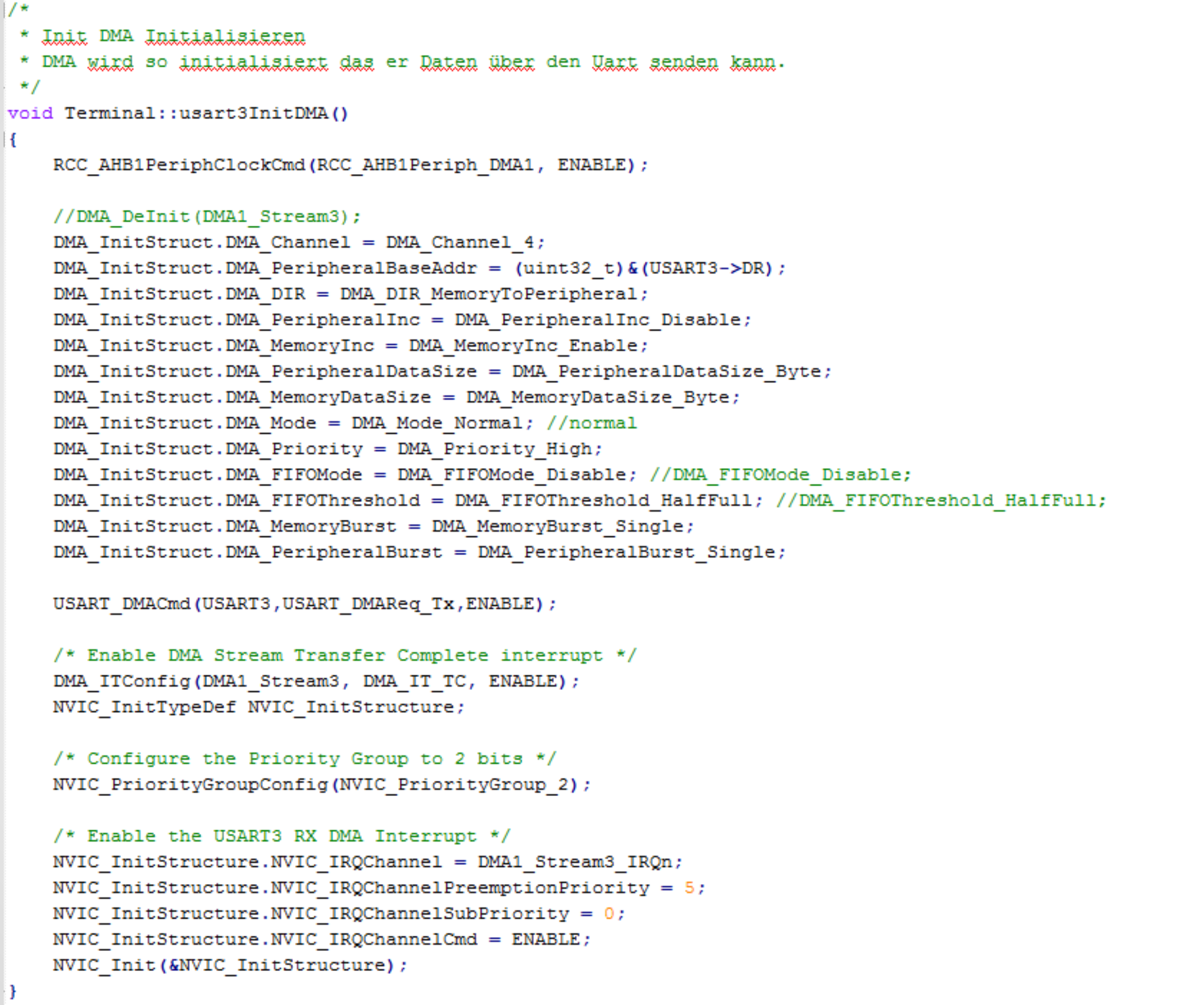
DMA Controller sendet Bytes über UART

Vorteile

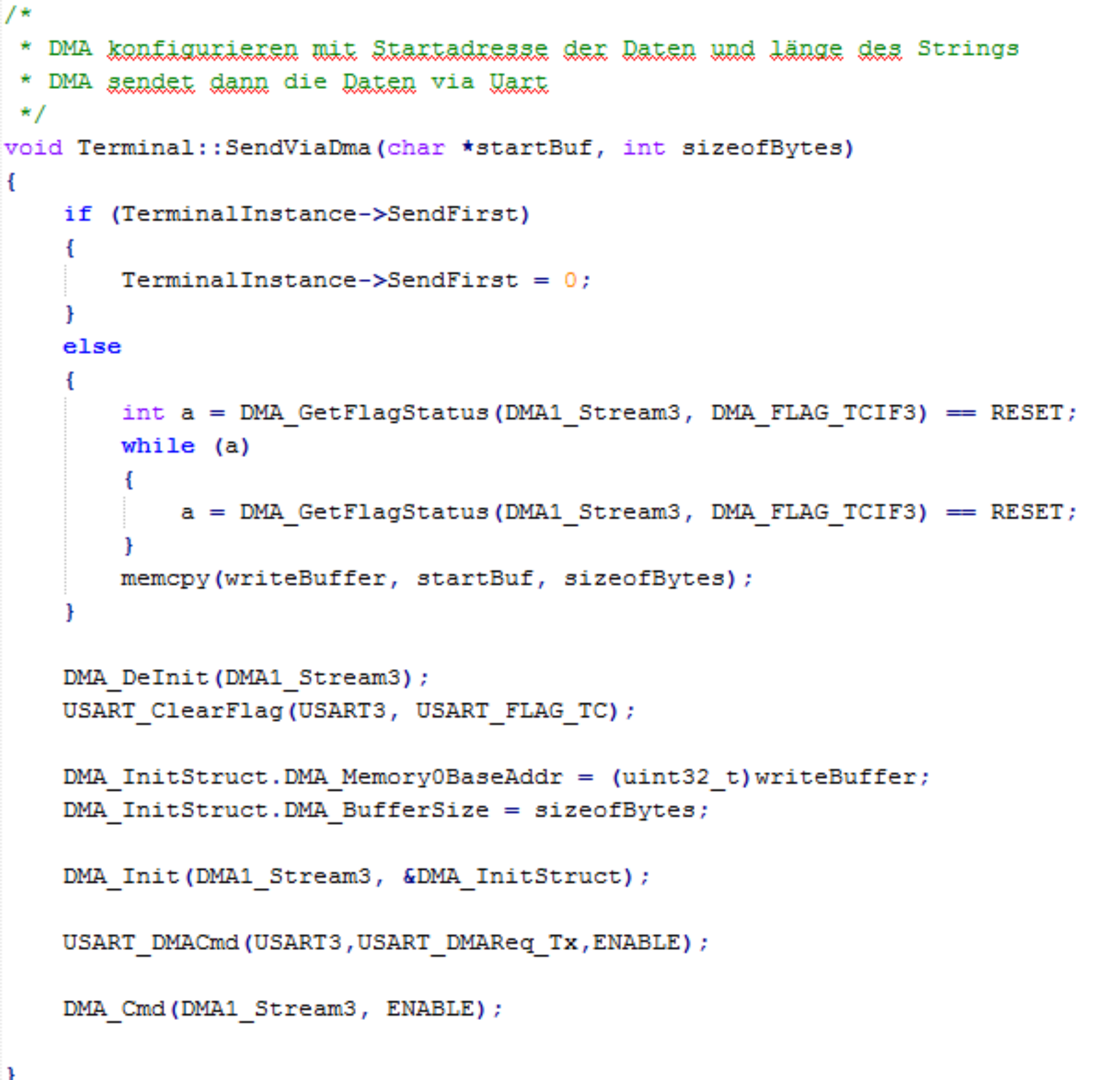
- CPU wird nicht vom langsamen UART ausgebremst

Nachteile

- Senden erst möglich nachdem DMA fertig
- Erhöhter aufwand im Programm



```
/*
 * Init DMA Initialisieren
 * DMA wird so initialisiert das er Daten über den Uart senden kann.
 */
void Terminal::usart3InitDMA()
{
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA1, ENABLE);

    //DMA_DeInit(DMA1_Stream3);
    DMA_InitStruct.DMA_Channel = DMA_Channel_4;
    DMA_InitStruct.DMA_PeripheralBaseAddr = (uint32_t)&(USART3->DR);
    DMA_InitStruct.DMA_DIR = DMA_DIR_MemoryToPeripheral;
    DMA_InitStruct.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    DMA_InitStruct.DMA_MemoryInc = DMA_MemoryInc_Enable;
    DMA_InitStruct.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
    DMA_InitStruct.DMA_MemoryDataSize = DMA_MemoryDataSize_Byte;
    DMA_InitStruct.DMA_Mode = DMA_Mode_Normal; //normal
    DMA_InitStruct.DMA_Priority = DMA_Priority_High;
    DMA_InitStruct.DMA_FIFOMode = DMA_FIFOMode_Disable; //DMA_FIFOMode_Disable;
    DMA_InitStruct.DMA_FIFOThreshold = DMA_FIFOThreshold_HalfFull; //DMA_FIFOThreshold_HalfFull;
    DMA_InitStruct.DMA_MemoryBurst = DMA_MemoryBurst_Single;
    DMA_InitStruct.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;

    USART_DMACmd(USART3,USART_DMAReq_Tx,ENABLE);

    /* Enable DMA Stream Transfer Complete interrupt */
    DMA_ITConfig(DMA1_Stream3, DMA_IT_TC, ENABLE);
    NVIC_InitTypeDef NVIC_InitStructure;

    /* Configure the Priority Group to 2 bits */
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);

    /* Enable the USART3 RX DMA Interrupt */
    NVIC_InitStructure.NVIC_IRQChannel = DMA1_Stream3_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 5;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```



```
/*
 * DMA konfigurieren mit Startadresse der Daten und länge des Strings
 * DMA sendet dann die Daten via Uart
 */
void Terminal::SendViaDma(char *startBuf, int sizeofBytes)
{
    if (TerminalInstance->SendFirst)
    {
        TerminalInstance->SendFirst = 0;
    }
    else
    {
        int a = DMA_GetFlagStatus(DMA1_Stream3, DMA_FLAG_TCIF3) == RESET;
        while (a)
        {
            a = DMA_GetFlagStatus(DMA1_Stream3, DMA_FLAG_TCIF3) == RESET;
        }
        memcpy(writeBuffer, startBuf, sizeofBytes);
    }

    DMA_DeInit(DMA1_Stream3);
    USART_ClearFlag(USART3, USART_FLAG_TC);

    DMA_InitStruct.DMA_Memory0BaseAddr = (uint32_t)writeBuffer;
    DMA_InitStruct.DMA_BufferSize = sizeofBytes;

    DMA_Init(DMA1_Stream3, &DMA_InitStruct);

    USART_DMACmd(USART3,USART_DMAReq_Tx,ENABLE);

    DMA_Cmd(DMA1_Stream3, ENABLE);

}
```

DMA und ADC

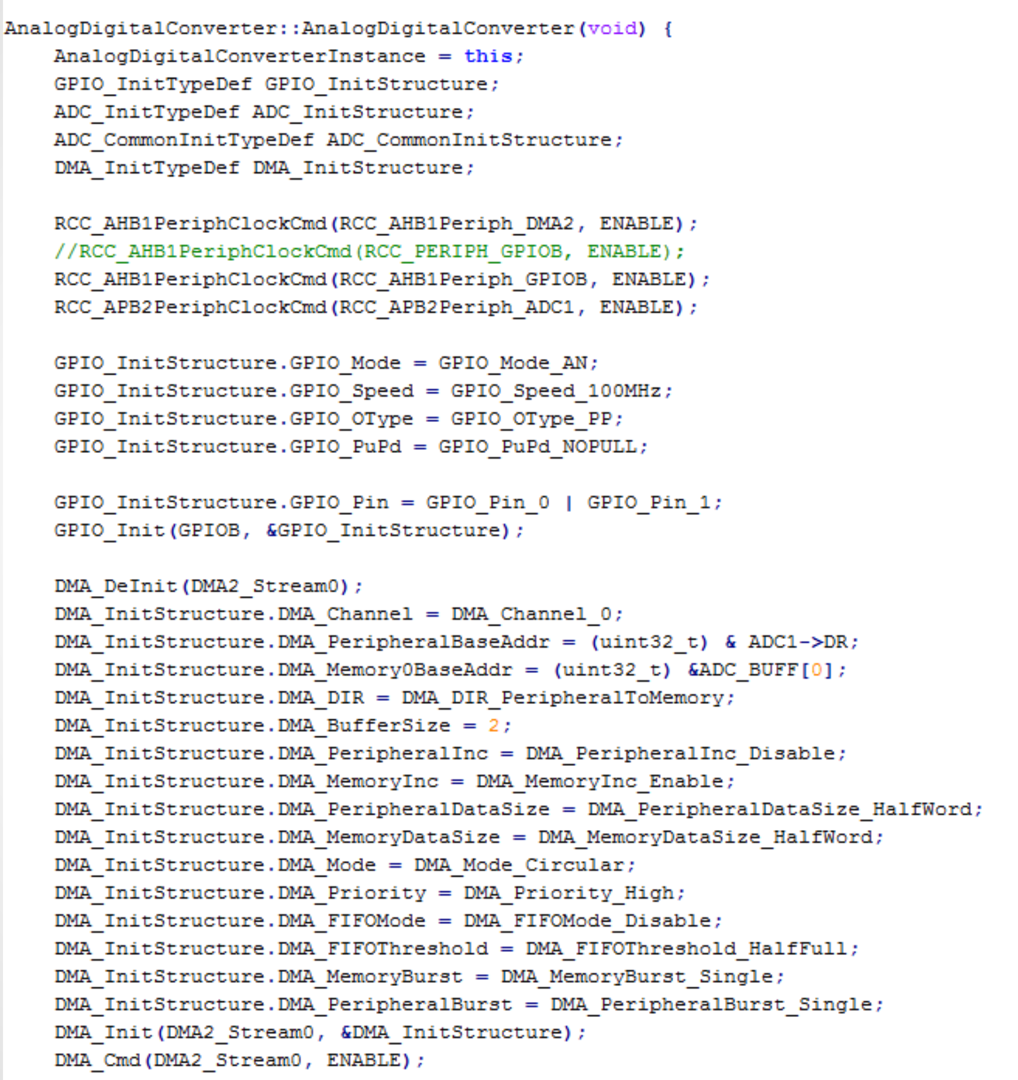
DMA Controller holt Werte vom ADC

Vorteile

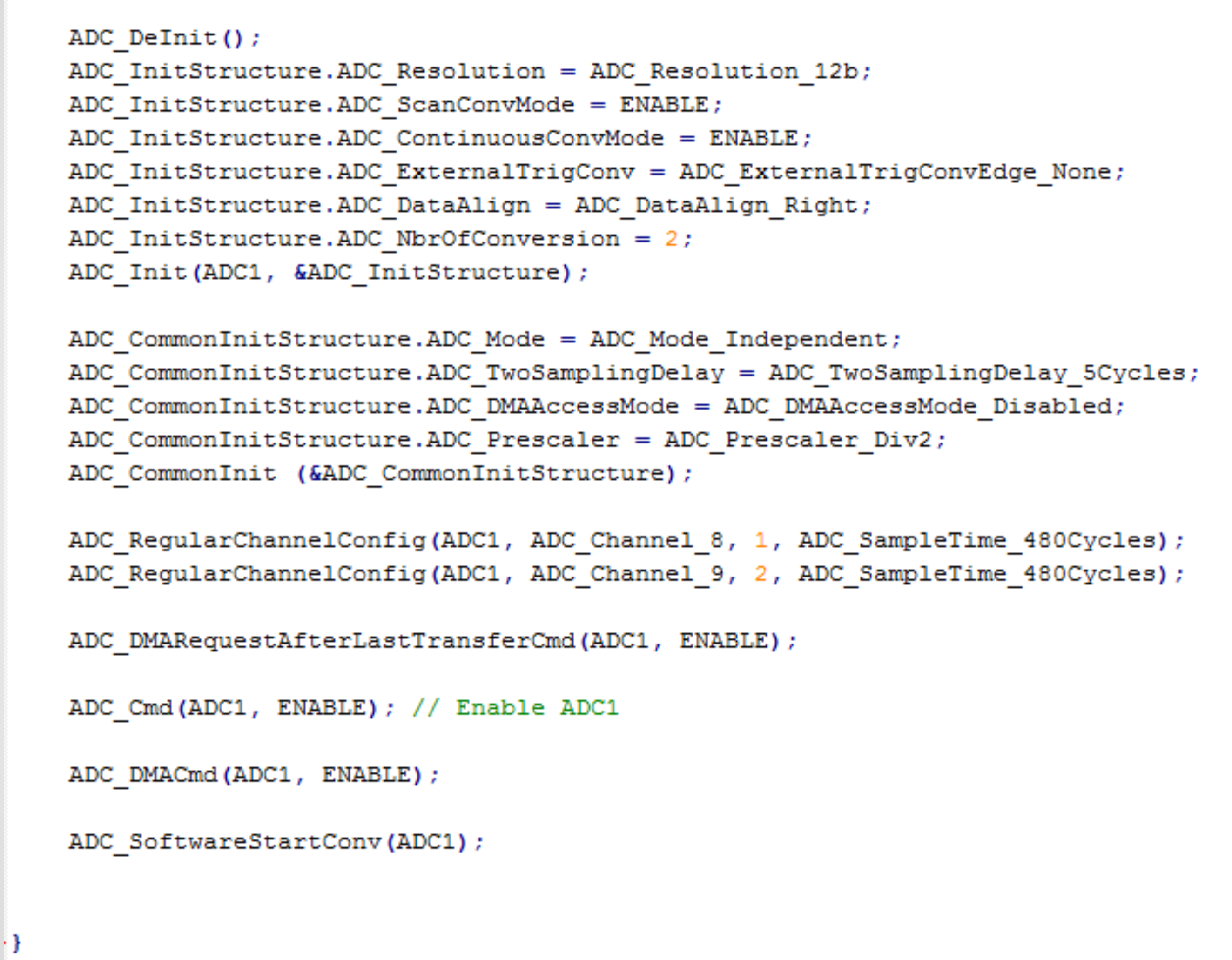
- CPU wird nicht vom langsamen ADC ausgebremst

Nachteile

- erhöhter Programmieraufwand



```
AnalogDigitalConverter::AnalogDigitalConverter(void) {
    AnalogDigitalConverterInstance = this;
    GPIO_InitTypeDef GPIO_InitStructure;
    ADC_InitTypeDef ADC_InitStructure;
    ADC_CommonInitTypeDef ADC_CommonInitStructure;
    DMA_InitTypeDef DMA_InitStructure;

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA2, ENABLE);
    //RCC_AHB1PeriphClockCmd(RCC_PERIPH_GPIOB, ENABLE);
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1;
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    DMA_DeInit(DMA2_Stream0);
    DMA_InitStructure.DMA_Channel = DMA_Channel_0;
    DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t) & ADC1->DR;
    DMA_InitStructure.DMA_Memory0BaseAddr = (uint32_t) &ADC_BUFF[0];
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralToMemory;
    DMA_InitStructure.DMA_BufferSize = 2;
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
    DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
    DMA_InitStructure.DMA_Priority = DMA_Priority_High;
    DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Disable;
    DMA_InitStructure.DMA_FIFOThreshold = DMA_FIFOThreshold_HalfFull;
    DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single;
    DMA_InitStructure.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
    DMA_Init(DMA2_Stream0, &DMA_InitStructure);
    DMA_Cmd(DMA2_Stream0, ENABLE);
```



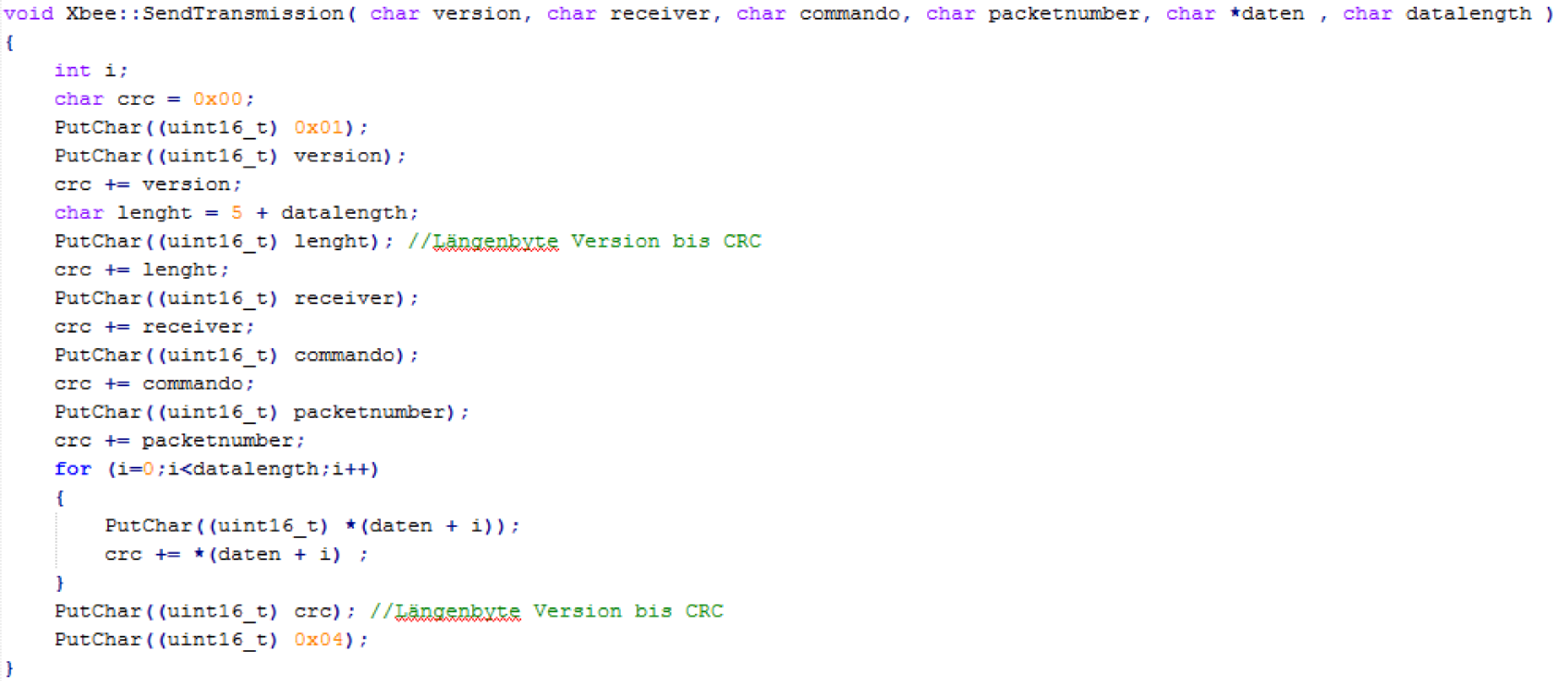
```
    ADC_DeInit();
    ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
    ADC_InitStructure.ADC_ScanConvMode = ENABLE;
    ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConvEdge_None;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_InitStructure.ADC_NbrOfConversion = 2;
    ADC_Init(ADC1, &ADC_InitStructure);

    ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;
    ADC_CommonInitStructure.ADC_TwoSamplingDelay = ADC_TwoSamplingDelay_5Cycles;
    ADC_CommonInitStructure.ADC_DMAAccessMode = ADC_DMAAccessMode_Disabled;
    ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div2;
    ADC_CommonInit (&ADC_CommonInitStructure);

    ADC_RegularChannelConfig(ADC1, ADC_Channel_8, 1, ADC_SampleTime_480Cycles);
    ADC_RegularChannelConfig(ADC1, ADC_Channel_9, 2, ADC_SampleTime_480Cycles);

    ADC_DMARequestAfterLastTransferCmd(ADC1, ENABLE);

    ADC_Cmd(ADC1, ENABLE); // Enable ADC1

    ADC_DMACmd(ADC1, ENABLE);

    ADC_SoftwareStartConv(ADC1);


}
```

Xbee

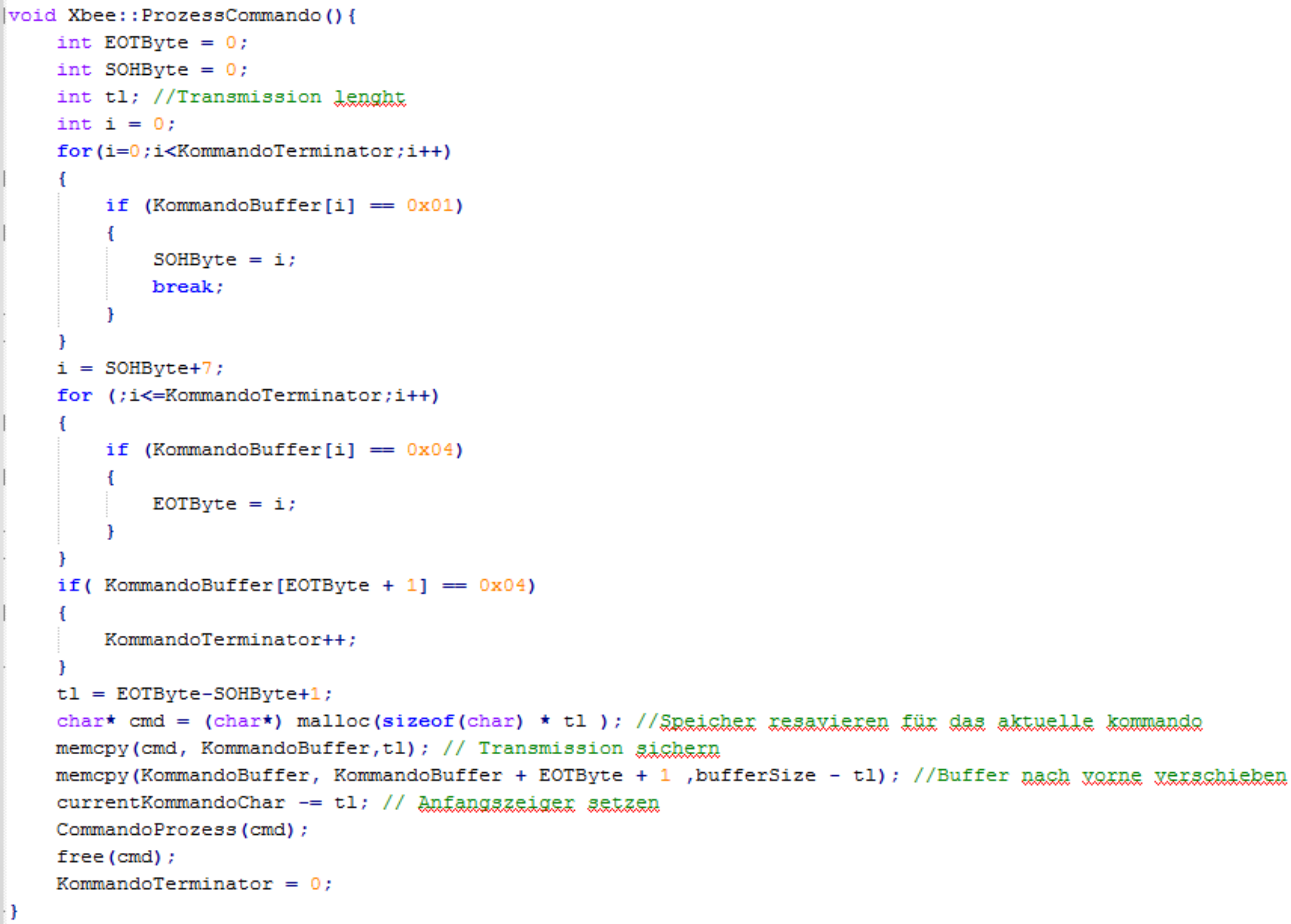
- Xbee arbeitet wie ein eine Serielle Schnittstelle
- Protokoll entworfen

Xbee

Byte	Wert	Beschreibung
0x00	0x01	SOH - Start
0x01	0x01	Protokollversionsnummer
0x02	0x__	Länge (Versionsnummer bis Ende Payload)
0x03	0x__	Empfänger: CORE(1),REMOUTE(2),GATEWAY(3)
0x04		Commando
0x05		Paketnummer
0x06 - FD		Payload
0x07 - FE		Checksumme (Summe von Version bis Ende Payload)
0x08 - FF	0x04	EOT - End of Transmission



```
void Xbee::SendTransmission( char version, char receiver, char commando, char packetnumber, char *daten , char datalength )
{
    int i;
    char crc = 0x00;
    PutChar((uint16_t) 0x01);
    PutChar((uint16_t) version);
    crc += version;
    char lenght = 5 + datalength;
    PutChar((uint16_t) lenght); //Längenbyte Version bis CRC
    crc += lenght;
    PutChar((uint16_t) receiver);
    crc += receiver;
    PutChar((uint16_t) commando);
    crc += commando;
    PutChar((uint16_t) packetnumber);
    crc += packetnumber;
    for (i=0;i<datalength;i++)
    {
        PutChar((uint16_t) *(daten + i));
        crc += *(daten + i) ;
    }
    PutChar((uint16_t) crc); //Längenbyte Version bis CRC
    PutChar((uint16_t) 0x04);
}
```

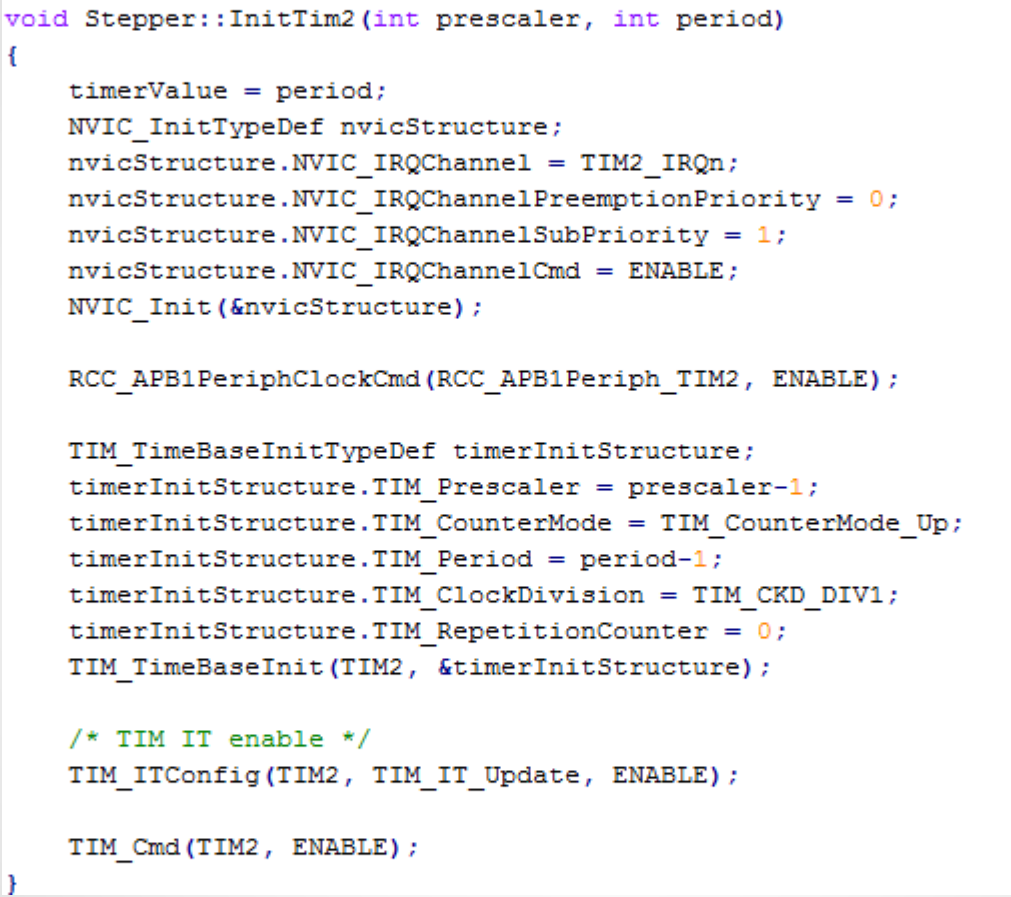


```
void Xbee::ProzessCommando(){
    int EOTByte = 0;
    int SOHByte = 0;
    int tl; //Transmission lenght
    int i = 0;
    for(i=0;i<KommandoTerminator;i++)
    {
        if (KommandoBuffer[i] == 0x01)
        {
            SOHByte = i;
            break;
        }
    }
    i = SOHByte+7;
    for (;i<=KommandoTerminator;i++)
    {
        if (KommandoBuffer[i] == 0x04)
        {
            EOTByte = i;
        }
    }
    if( KommandoBuffer[EOTByte + 1] == 0x04)
    {
        KommandoTerminator++;
    }
    tl = EOTByte-SOHByte+1;
    char* cmd = (char*) malloc(sizeof(char) * tl ); //Speicher resavieren für das aktuelle kommando
    memcpy(cmd, KommandoBuffer,tl); // Transmission sichern
    memcpy(KommandoBuffer, KommandoBuffer + EOTByte + 1 ,bufferSize - tl); //Buffer nach vorne verschieben
    currentKommandoChar -= tl; // Anfangszeiger setzen
    CommandoProzess(cmd);
    free(cmd);
    KommandoTerminator = 0;
}
```

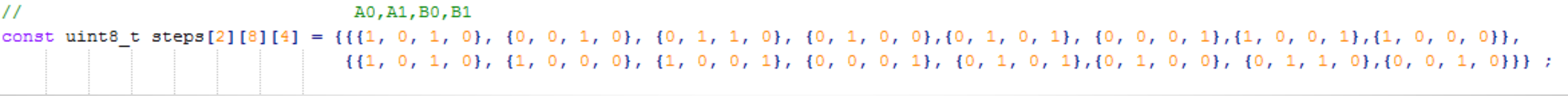
Stepper

Der Stepper soll nicht die CPU beeinflussen

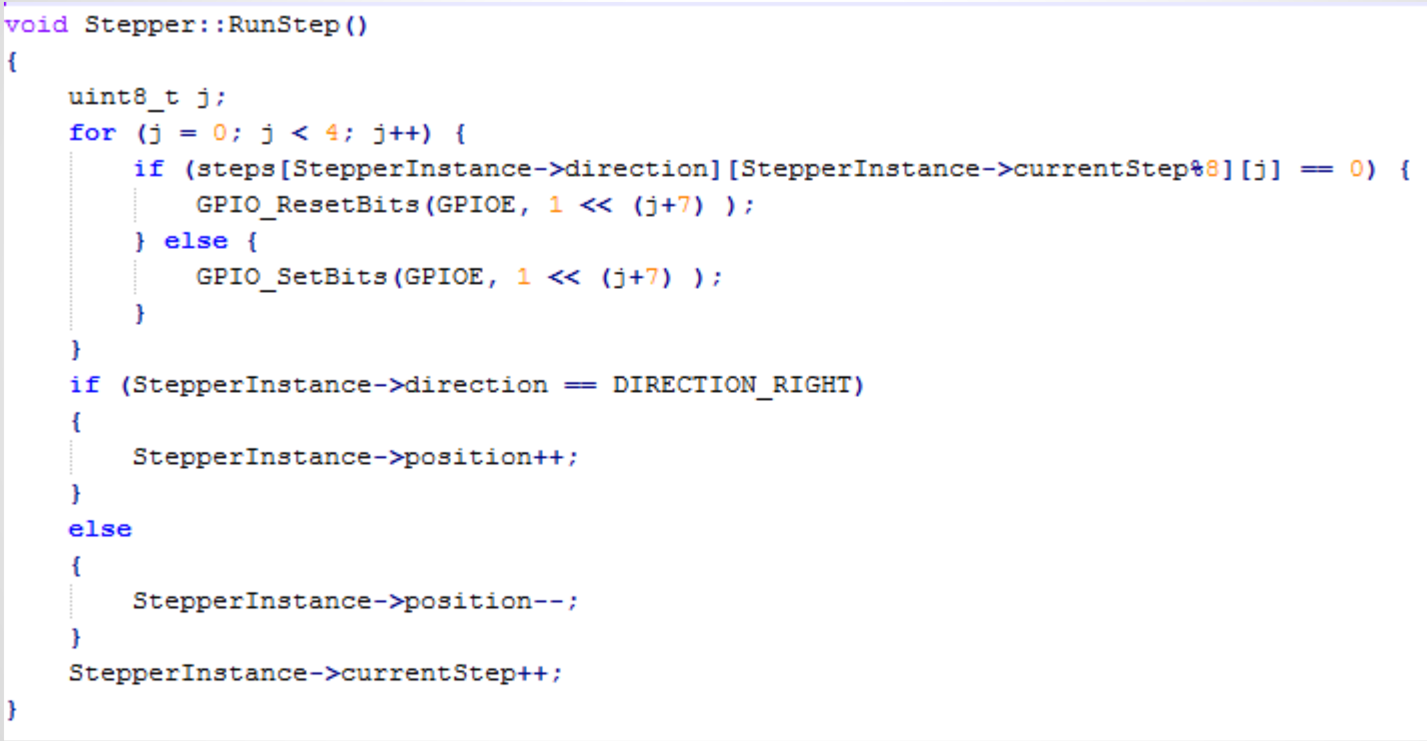
- Stepper via Timer ansteuern
- im Interrupt Sprung ausführen
- Anlauf- und Bremsvorgang mit einer Rampe



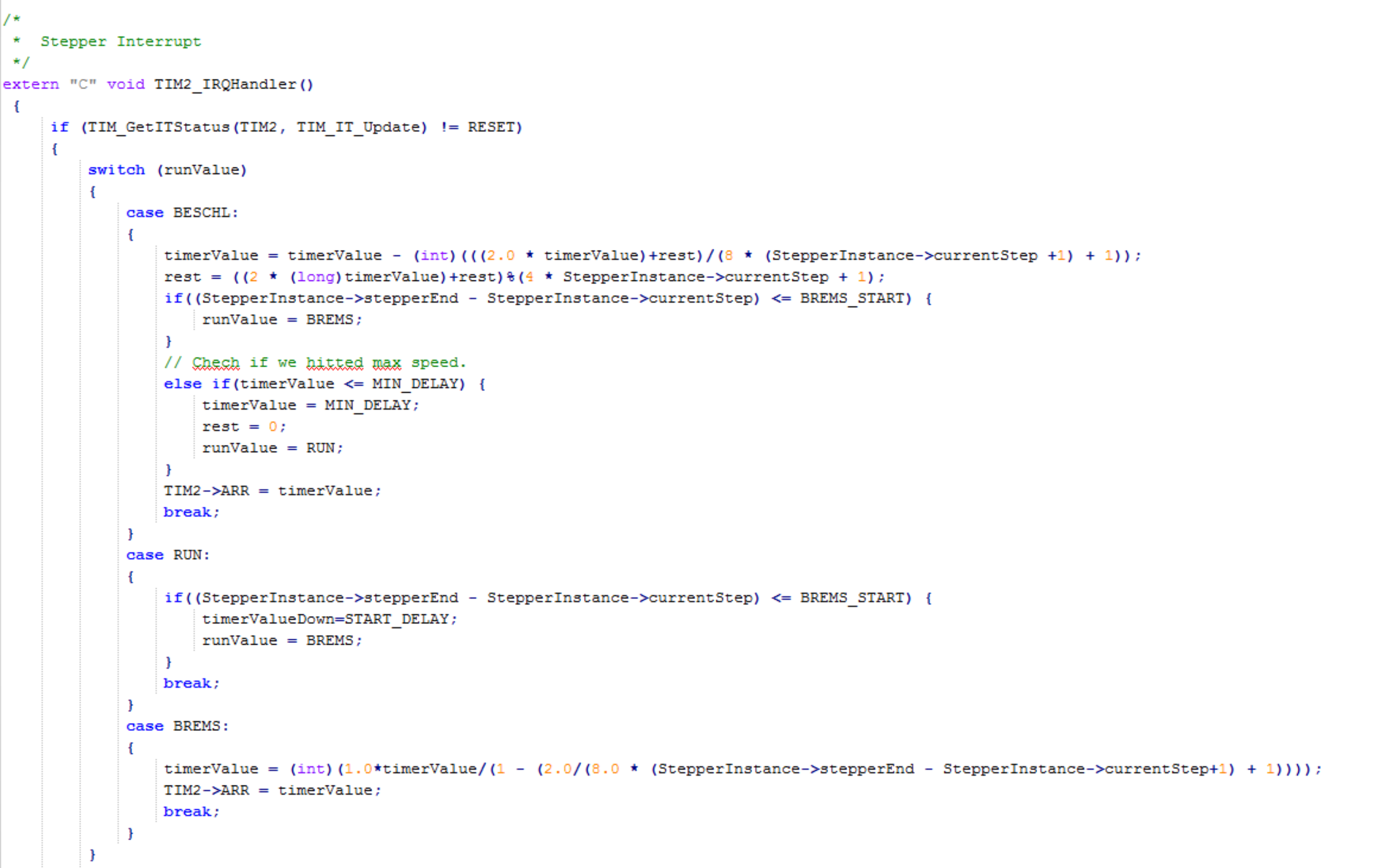
```
void Stepper::InitTim2(int prescaler, int period)
{
    timerValue = period;
    NVIC_InitTypeDef nvicStructure;
    nvicStructure.NVIC_IRQChannel = TIM2_IRQn;
    nvicStructure.NVIC_IRQChannelPreemptionPriority = 0;
    nvicStructure.NVIC_IRQChannelSubPriority = 1;
    nvicStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&nvicStructure);

    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

    TIM_TimeBaseInitTypeDef timerInitStructure;
    timerInitStructure.TIM_Prescaler = prescaler-1;
    timerInitStructure.TIM_CounterMode = TIM_CounterMode_Up;
    timerInitStructure.TIM_Period = period-1;
    timerInitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
    timerInitStructure.TIM_RepetitionCounter = 0;
    TIM_TimeBaseInit(TIM2, &timerInitStructure);

    /* TIM IT enable */
    TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);

    TIM_Cmd(TIM2, ENABLE);
}
```



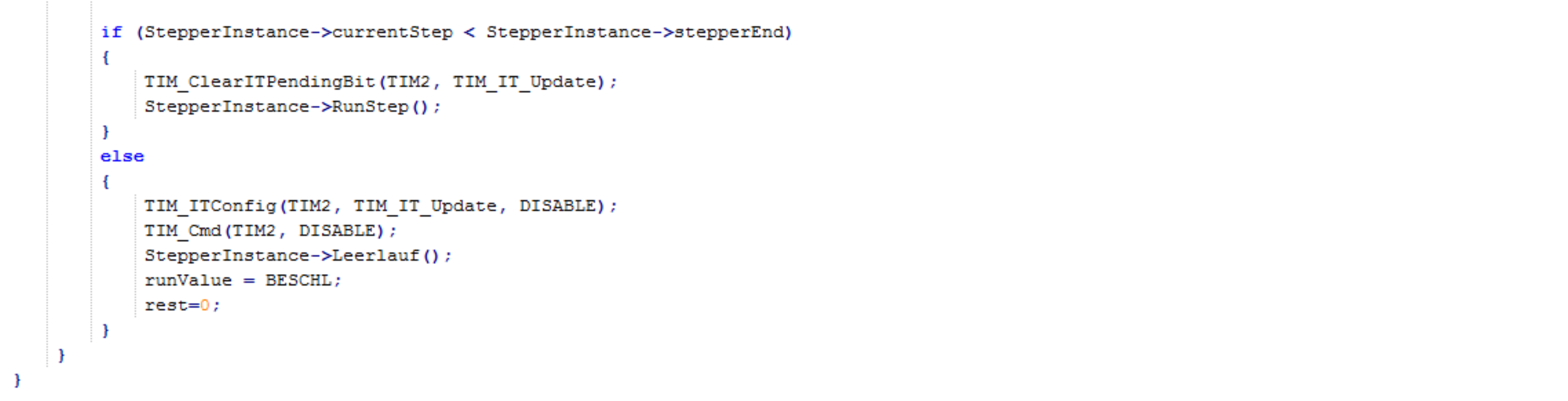
```
//                                  A0,A1,B0,B1
const uint8_t steps[2][8][4] = {{{1, 0, 1, 0}, {0, 0, 1, 0}, {0, 1, 1, 0}, {0, 1, 0, 0},{0, 1, 0, 1}, {0, 0, 0, 1},{1, 0, 0, 1},{1, 0, 0, 0}},
                                 {{1, 0, 1, 0}, {1, 0, 0, 0}, {1, 0, 0, 1}, {0, 0, 0, 1}, {0, 1, 0, 1},{0, 1, 0, 0}, {0, 1, 1, 0},{0, 0, 1, 0}}} ;
```



```
void Stepper::RunStep()
{
    uint8_t j;
    for (j = 0; j < 4; j++) {
        if (steps[StepperInstance->direction][StepperInstance->currentStep%8][j] == 0) {
            GPIO_ResetBits(GPIOE, 1 << (j+7) );
        } else {
            GPIO_SetBits(GPIOE, 1 << (j+7) );
        }
    }
    if (StepperInstance->direction == DIRECTION_RIGHT)
    {
        StepperInstance->position++;
    }
    else
    {
        StepperInstance->position--;
    }
    StepperInstance->currentStep++;
}
```



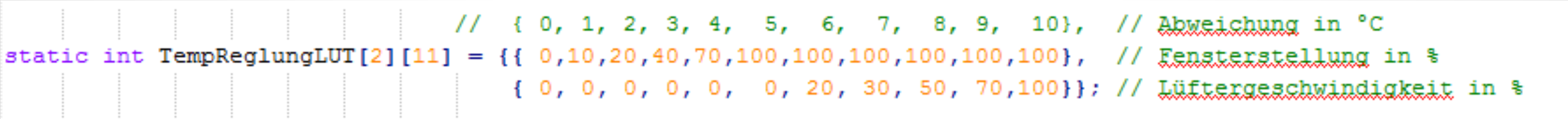
```
/*
 *  Stepper Interrupt
 */
extern "C" void TIM2_IRQHandler()
 {
    if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET)
    {
        switch (runValue)
        {
            case BESCHL:
            {
                timerValue = timerValue - (int)(((2.0 * timerValue)+rest)/(8 * (StepperInstance->currentStep +1) + 1));
                rest = ((2 * (long)timerValue)+rest)%(4 * StepperInstance->currentStep + 1);
                if((StepperInstance->stepperEnd - StepperInstance->currentStep) <= BREMS_START) {
                    runValue = BREMS;
                }
                // Chech if we hitted max speed.
                else if(timerValue <= MIN_DELAY) {
                    timerValue = MIN_DELAY;
                    rest = 0;
                    runValue = RUN;
                }
                TIM2->ARR = timerValue;
                break;
            }
            case RUN:
            {
                if((StepperInstance->stepperEnd - StepperInstance->currentStep) <= BREMS_START) {
                    timerValueDown=START_DELAY;
                    runValue = BREMS;
                }
                break;
            }
            case BREMS:
            {
                timerValue = (int)(1.0*timerValue/(1 - (2.0/(8.0 * (StepperInstance->stepperEnd - StepperInstance->currentStep+1) + 1))));
                TIM2->ARR = timerValue;
                break;
            }
        }
```



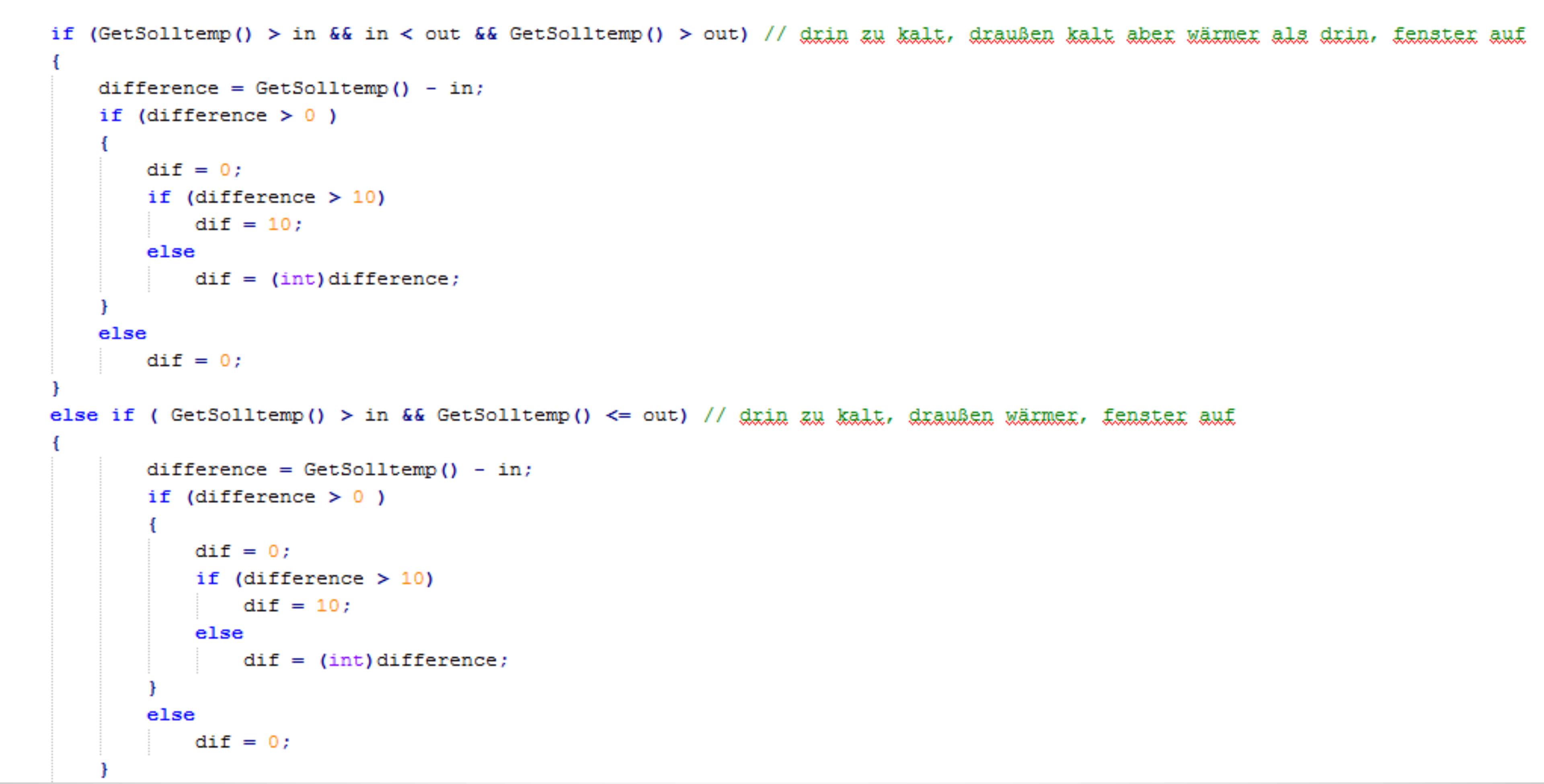
```
        if (StepperInstance->currentStep < StepperInstance->stepperEnd)
        {
            TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
            StepperInstance->RunStep();
        }
        else
        {
            TIM_ITConfig(TIM2, TIM_IT_Update, DISABLE);
            TIM_Cmd(TIM2, DISABLE);
            StepperInstance->Leerlauf();
            runValue = BESCHL;
            rest=0;
        }
    }
}
```

Fenstersteuerung

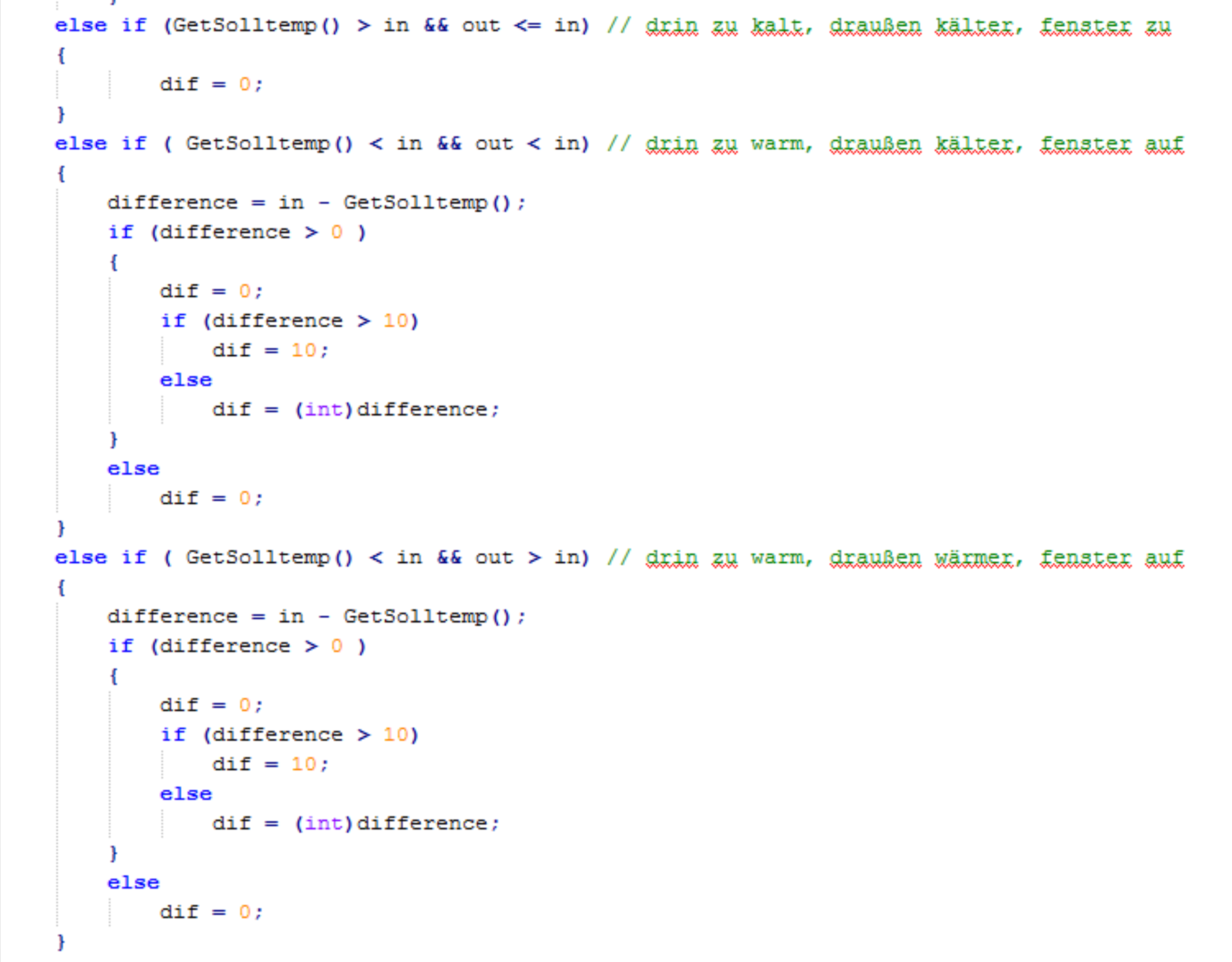
Realisiert als Lookuptable



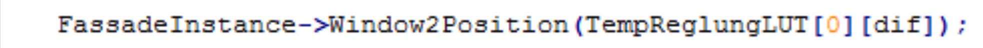
```
                                //  { 0, 1, 2, 3, 4,  5,  6,  7,  8, 9,  10},  // Abweichung in °C
static int TempReglungLUT[2][11] = {{ 0,10,20,40,70,100,100,100,100,100,100},  // Fensterstellung in %
                                    { 0, 0, 0, 0, 0,  0, 20, 30, 50, 70,100}}; // Lüftergeschwindigkeit in %
```



```
if (GetSolltemp() > in && in < out && GetSolltemp() > out) // drin zu kalt, draußen kalt aber wärmer als drin, fenster auf
{
    difference = GetSolltemp() - in;
    if (difference > 0 )
    {
        dif = 0;
        if (difference > 10)
            dif = 10;
        else
            dif = (int)difference;
    }
    else
        dif = 0;
}
else if ( GetSolltemp() > in && GetSolltemp() <= out) // drin zu kalt, draußen wärmer, fenster auf
{
        difference = GetSolltemp() - in;
        if (difference > 0 )
        {
            dif = 0;
            if (difference > 10)
                dif = 10;
            else
                dif = (int)difference;
        }
        else
            dif = 0;
    }
```



```
}
else if (GetSolltemp() > in && out <= in) // drin zu kalt, draußen kälter, fenster zu
{
        dif = 0;
}
else if ( GetSolltemp() < in && out < in) // drin zu warm, draußen kälter, fenster auf
{
    difference = in - GetSolltemp();
    if (difference > 0 )
    {
        dif = 0;
        if (difference > 10)
            dif = 10;
        else
            dif = (int)difference;
    }
    else
        dif = 0;
}
else if ( GetSolltemp() < in && out > in) // drin zu warm, draußen wärmer, fenster auf
{
    difference = in - GetSolltemp();
    if (difference > 0 )
    {
        dif = 0;
        if (difference > 10)
            dif = 10;
        else
            dif = (int)difference;
    }
    else
        dif = 0;
}
```



```
FassadeInstance->Window2Position(TempReglungLUT[0][dif]);
```

Fragen?