```
 84  /*****************************************************************************/
 85  /*!
 86   *  \fn     int main (void)
 87   *
 88   *  \brief  main
 89   *
 90  */
 91  /*****************************************************************************/
 92  int main (void)
 93  {
 94    // Variablen
 95
 96    // Initialisierung System
 97    SetSystemClock();
 98    SysTickTimer_Init();
 99
100    // Initialisierung Peripherie
101    GPIO_Init();
102    TIM1_Init();
103    ADC_Init();
104
105    // Start Peripherie
106    TIM1_SetCOM();    // Kommutierungsevent
107
108    // Endlosschleife
109    while (1)
110    {
111
112    }
113  }// ENDE Funktion
```

```
  66  /*****************************************************************************/
  67  /*!
  68   *  \fn     void TIM1_Init (void)
  69   *
  70   *  \brief  Initialisiert den TIM1
  71   *
  72   *
  73  */
  74  /*****************************************************************************/
  75  void TIM1_Init (void)
  76  {
  77    // Timer1 clock aktivieren
  78    TIM1_ClockEnable();
  79
  80    // Clock Prescal-Counter: CLK_CNT = 48/2 = 24 MHz
  81    TIM1->PSC = 0x0001;
  82
  83    // PWM - Frequenz : 10kHz
  84    TIM1->ARR = 2399;
  85
  86    // Voreinstellung Duty-Cycle
  87    TIM1->CCR1 = 1199;                  // DutyCycle 50%
  88    TIM1->CCR2 = 1199;                  //    -- " --
  89    TIM1->CCR3 = 1199;                  //    -- " --
  90    TIM1->CCR4 = (TIM1->ARR)/2;         // Zeitpunkt fuer AD-Trigger
  91
  92    // Konfiguration
  93    TIM1->CR1   &=~ TIM_CR1_DIR;        // Counter zählt hoch
  94    TIM1->CR1   &=~ TIM_CR1_CMS;        // Edge-aligned mode
  95    TIM1->CR1   &=~ TIM_CR1_CKD;        // Clock Totzeit ist gleich wie Eingangstakt
  96    TIM1->CR1   &=~ TIM_CR1_OPM;        // Counter wird bei Update-Event nicht gestoppt
  97    TIM1->CR1   &=~ TIM_CR1_URS;        // Update kann durch Software angefordert werden
  98    TIM1->CR1   &=~ TIM_CR1_UDIS;       // Update enabled
  99
 100    TIM1->CR2   &=~ TIM_CR2_OIS1;       // Leerlaufzustand CH1  = 0 (wenn MOE = 0)
 101    TIM1->CR2   &=~ TIM_CR2_OIS1N;      // Leerlaufzustand CH1N = 0 (wenn MOE = 0)
 102    TIM1->CR2   &=~ TIM_CR2_OIS2;       // Leerlaufzustand CH2  = 0 (wenn MOE = 0)
 103    TIM1->CR2   &=~ TIM_CR2_OIS2N;      // Leerlaufzustand CH2N = 0 (wenn MOE = 0)
 104    TIM1->CR2   &=~ TIM_CR2_OIS3;       // Leerlaufzustand CH3  = 0 (wenn MOE = 0)
 105    TIM1->CR2   &=~ TIM_CR2_OIS3N;      // Leerlaufzustand CH3N = 0 (wenn MOE = 0)
 106    TIM1->CR2   &=~ TIM_CR2_OIS4;       // Leerlaufzustand CH4  = 0 (wenn MOE = 0)
 107    TIM1->CR2   &=~ TIM_CR2_CCUS;       // Update erfolgt nur ueber COMG
 108    TIM1->CR2   |=  TIM_CR2_CCPC;       // CCxE,CCxNE,OCxM werden vorgeladen und koennen
 109                                        // dann ueber COMG aktualisiert werden
 110
 111    TIM1->CCMR1 |=  TIM_CCMR1_OC1PE;    // OC1 preload enable
 112    TIM1->CCMR1 |=  TIM_CCMR1_OC1M_1    // PWM-Mode 1 (Left-aligned)
 113                |   TIM_CCMR1_OC1M_2;   //         --- " ----
 114    TIM1->CCMR1 |=  TIM_CCMR1_OC2PE;    // OC2 preload enable
 115    TIM1->CCMR1 |=  TIM_CCMR1_OC2M_1    // PWM-Mode 1 (Left-aligned)
 116                |   TIM_CCMR1_OC2M_2;   //         --- " ----
 117    TIM1->CCMR2 |=  TIM_CCMR2_OC3PE;    // OC3 preload enable
 118    TIM1->CCMR2 |=  TIM_CCMR2_OC3M_1    // PWM-Mode 1 (Left aligned)
 119                |   TIM_CCMR2_OC3M_2;   //         --- " ----
 120    TIM1->CCMR2 &=~ TIM_CCMR2_CC4S;     // OC4 als Output
 121    TIM1->CCMR2 |=  TIM_CCMR2_OC4PE;    // OC4 preload enable
 122    TIM1->CCMR2 &=~ TIM_CCMR2_OC4M;     // OC4 Frozen : Kein Signal auf der Hardware
 123                                        // Es erfolgt nur Vergleich von CNT und CCR4
 124
 125    TIM1->CCER  |=  TIM_CCER_CC1E;      // Capture/Compare Channel 1 enable
 126    TIM1->CCER  &=~ TIM_CCER_CC1P;      // Channel 1 als active high konfiguriert
 127    TIM1->CCER  |=  TIM_CCER_CC1NE;     // Capture/Compare Channel 1N enable
 128    TIM1->CCER  &=~ TIM_CCER_CC1NP;     // Channel 1N als active high konfiguriert
 129    TIM1->CCER  |=  TIM_CCER_CC2E;      // Capture/Compare Channel 2 enable
 130    TIM1->CCER  &=~ TIM_CCER_CC2P;      // Channel 2 als active high konfiguriert
 131    TIM1->CCER  |=  TIM_CCER_CC2NE;     // Capture/Compare Channel 2N enable
 132    TIM1->CCER  &=~ TIM_CCER_CC2NP;     // Channel 2N als active high konfiguriert
 133    TIM1->CCER  |=  TIM_CCER_CC3E;      // Capture/Compare Channel 3 enable
 134    TIM1->CCER  &=~ TIM_CCER_CC3P;      // Channel 3 als active high konfiguriert
 135    TIM1->CCER  |=  TIM_CCER_CC3NE;     // Capture/Compare Channel 3N enable
 136    TIM1->CCER  &=~ TIM_CCER_CC3NP;     // Channel 3N als active high konfiguriert
 137    TIM1->CCER  |=  TIM_CCER_CC4E;      // Capture/Compare Channel 4 enable
 138    TIM1->CCER  &=~ TIM_CCER_CC4P;      // Channel 4 als active high konfiguriert
 139
 140    // DIER_CC4IE
 141  //  TIM1->DIER  |=  TIM_DIER_CC4IE;     // Interrupt bei Capture Ereignis aktiviert
 142
 143  //  TIM1->BDTR |=  TIM_BDTR_AOE;        // MOE kann automatisch beim nächsten Event gesetzt werden
```

```
144     TIM1->BDTR |=  TIM_BDTR_OSSR;       // Outputs enable
145     TIM1->BDTR &=~ TIM_BDTR_OSSI;       // Output disable wenn MOE=0
146     TIM1->BDTR &=~ TIM_BDTR_DTG;        // Totzeit 1us
147     TIM1->BDTR |=  TIM_BDTR_DTG_4       // Einstellung für Totzeit bei f(TIM1)=24MHz
148                |   TIM_BDTR_DTG_5;      //            -- " --
149
150     // Start TIM1
151     TIM1_Start();
152
153     // Master Output enable
154     TIM1->BDTR |=  TIM_BDTR_MOE;        // Master Output enable
155
156     // Interrupt aktivieren
157   //  IRQ_TIM1_CC_Init();
158
159   }// ENDE Funktion
```

```
  62  /*****************************************************************************/
  63  /*!
  64   *  \fn     void ADC_Init (void)
  65   *
  66   *  \brief  Initialisierung des AD-Wandlers
  67   *
  68   *
  69  */
  70  /*****************************************************************************/
  71  void ADC_Init(void)
  72  {
  73    // ADC Clock Mode
  74    ADC1->CFGR2   &=~ ADC_CFGR2_CKMODE;   // ADCCLK Asynchronous Clock= 14MHz
  75
  76    // ADC-Clock aktivieren
  77    ADC_ClockEnable();
  78
  79    // ADC-Konfiguration
  80    ADC1->IER     |=  ADC_IER_EOCIE;      // Interrupt nach jeder beendeten Wandlung
  81    ADC1->CFGR1   &=~ ADC_CFGR1_DISCEN;   // Discontinous Mode deaktiviert
  82    ADC1->CFGR1   &=~ ADC_CFGR1_AUTOFF;   // Auto-off mode deaktiviert
  83    ADC1->CFGR1   &=~ ADC_CFGR1_WAIT;     // Wait mode deaktiviert
  84    ADC1->CFGR1   &=~ ADC_CFGR1_CONT;     // Single conversion mode
  85    ADC1->CFGR1   |=  ADC_CFGR1_OVRMOD;   // Bei Overrun wird Datenregister mit neuem Wert überschrieben
  86    ADC1->CFGR1   &=~ ADC_CFGR1_ALIGN;    // Right alignment
  87    ADC1->CFGR1   &=~ ADC_CFGR1_RES;      // Aufloesung: 12-Bit
  88    ADC1->CFGR1   &=~ ADC_CFGR1_SCANDIR;  // Scan-Sequenz aufwärts von Ch0 -> Ch17
  89    ADC1->CFGR1   &=~ ADC_CFGR1_DMAEN;    // DMA disabled
  90  //  ADC1->CFGR1   &=~ ADC_CFGR1_EXTEN;    // Hardware-Trigger deaktiviert, Start per Software
  91  //  ADC1->CFGR1   |=  ADC_CFGR1_EXTEN;    // Hardware-Trigger bei steigender und fallender Flanke
  92    ADC1->CFGR1   |=  ADC_CFGR1_EXTEN_0;  // Hardware-Trigger bei steigender Flanke
  93    ADC1->CFGR1   |=  ADC_CFGR1_EXTSEL_0; // Trigger-Auswahl TRG1: TIM1_CC4
  94
  95    ADC1->SMPR    &=~ ADC_SMPR_SMP;       // 1.5 ADC-clock cycles sample time
  96
  97    // Kanalauswahl
  98    // !! Achtung bei Auskommentierung !!
  99    // !! Wird ein Kanal auskommentiert, so muss die Anzahl der nun vorgenommen Wandlungen
 100    // !! im Handler des ADC beruecksichtigt werden, da sonst ein Ergebnis einem
 101    // !! falschen Kanal zugewiesen werden kann
 102  //  ADC1->CHSELR  |=  ADC_CHSELR_CHSEL0;  // Wandlung auf Kanal 0: Strom_U1_Shunt
 103  //  ADC1->CHSELR  |=  ADC_CHSELR_CHSEL1;  // Wandlung auf Kanal 1: Strom V2
 104  //  ADC1->CHSELR  |=  ADC_CHSELR_CHSEL2;  // Wandlung auf Kanal 2: Strom W2
 105  //  ADC1->CHSELR  |=  ADC_CHSELR_CHSEL3;  // Wandlung auf Kanal 3: BEMF_U
 106  //  ADC1->CHSELR  |=  ADC_CHSELR_CHSEL4;  // Wandlung auf Kanal 4: BEMF_V
 107  //  !! ADC1->CHSELR |=  ADC_CHSELR_CHSEL5;  // Wandlung auf Kanal 5 nicht zulaessig!!
 108  //  ADC1->CHSELR  |=  ADC_CHSELR_CHSEL6;  // Wandlung auf Kanal 6: BEMF_W
 109  //  ADC1->CHSELR  |=  ADC_CHSELR_CHSEL7;  // Wandlung auf Kanal 7: Temp. Endstufe
 110  //  ADC1->CHSELR  |=  ADC_CHSELR_CHSEL8;  // Wandlung auf Kanal 8: Temp. Motor
 111    ADC1->CHSELR  |=  ADC_CHSELR_CHSEL9;  // Wandlung auf Kanal 9: Sollwert
 112  //  !! adc1->chselr |=  adc_chselr_chsel10; // wandlung auf kanal 10 nicht zulaessig!!
 113  //  !! adc1->chselr |=  adc_chselr_chsel11; // wandlung auf kanal 11 nicht zulaessig!!
 114  //  !! adc1->chselr |=  adc_chselr_chsel12; // wandlung auf kanal 12 nicht zulaessig!!
 115  //  !! adc1->chselr |=  adc_chselr_chsel13; // wandlung auf kanal 13 nicht zulaessig!!
 116  //  !! adc1->chselr |=  adc_chselr_chsel14; // wandlung auf kanal 14 nicht zulaessig!!
 117  //  !! adc1->chselr |=  adc_chselr_chsel15; // wandlung auf kanal 15 nicht zulaessig!!
 118  //  !! adc1->chselr |=  adc_chselr_chsel16; // wandlung auf kanal 16 nicht zulaessig!!
 119  //  !! adc1->chselr |=  adc_chselr_chsel17; // wandlung auf kanal 17 nicht zulaessig!!
 120  //  !! adc1->chselr |=  adc_chselr_chsel18; // wandlung auf kanal 18 nicht zulaessig!!
 121
 122    // Interrupt konfigurieren und freigeben
 123    IRQ_ADC_Init();
 124
 125    // ADC-Calibration
 126    ADC1->CR |= ADC_CR_ADCAL;
 127    while ((ADC1->CR & ADC_CR_ADCAL) == ADC_CR_ADCAL)   // Warten auf erfolgreiche Kalibrierung
 128    {}
 129
 130    // ADC einschalten
 131    ADC1->CR = ADC_CR_ADEN;
 132    // Warten auf AD-Ready
 133    while ((ADC1->ISR & ADC_ISR_ADRDY)==0)
 134    {}
 135
 136    ADC_Start();
 137
 138  }// ENDE Funktion
```