Frequenzumrichter / 3-Phasenbrücke

Motivation

Heutzutage gibt es für die verschiedensten Teilgebiete der Elektrotechnik bereits eine Vielzahl vorkonfigurierter, modularer und kostengünstiger Entwicklungsumgebungen, dank deren Verfügbarkeit ein Testsystem bzw. ein Prototyp für die unterschiedlichsten Anwendungen meist relativ schnell realisiert werden kann. Hierzu zählen beispielsweise Entwicklungsumgebungen aus folgenden Bereichen:

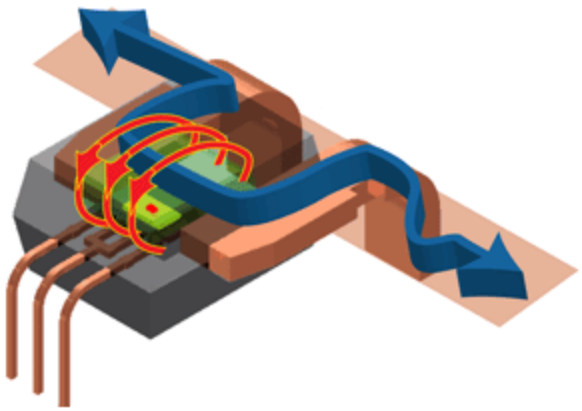
- **Mikrocontroller (STM, Microchip, Atmel, ...)**
- **FPGAs und DSPs (Altera, Xilinx, TI, Analog Devices, ...)**
- **Motorsysteme (Maxon Motors, Trinamic, Infineon, ...)**
- **Mess- und Sensortechnik (Analog Devices, Linear Technology, ...)**

Zielsetzung

Ziel dieses Projektes ist der Entwurf sowie die Fertigung einer möglichst universellen 3-Phasenbrücke. Im Vordergrund dieser Entwicklung sollen hierbei ein robustes Design, wie auch flexible Ansteuermöglichkeiten der Brücke stehen.

Stromsensoren

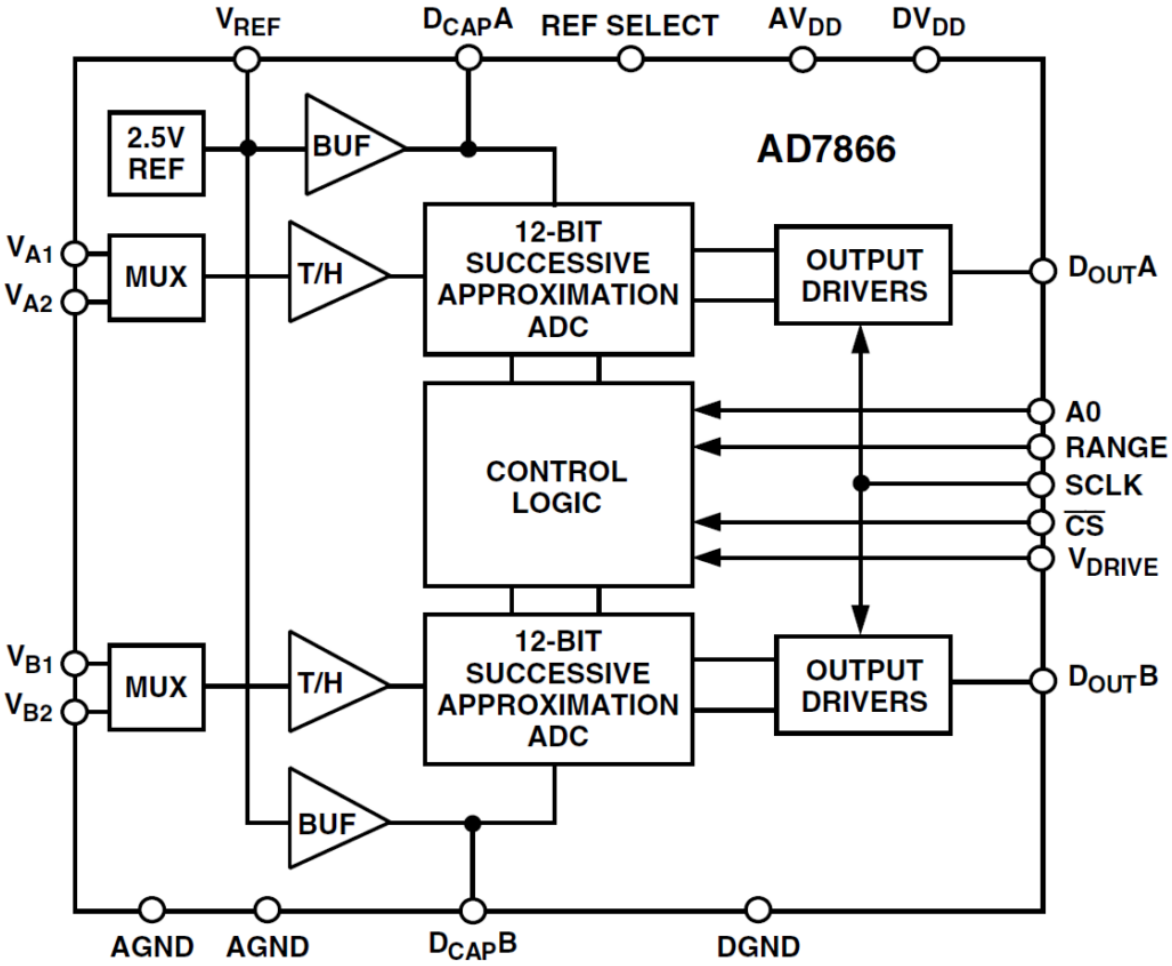
Zur Realisierung einer galvanisch getrennten Strommessung aller 3 Phasen wurden die Strom-Messwandler ACS770 (Allegro Micro Systems) ausgew□ahlt. Diese Sensoren basieren auf dem Halleffekt und sind pinkompatibel in den Messbereichen ± 50A, ± 100A, ± 150A und ± 200A verf□ügbar.
Die Wandler werden □über 5V versorgt und stellen eine stromproportionale Spannung am Ausgang zur Verf□ügung.



A/D-Wandler

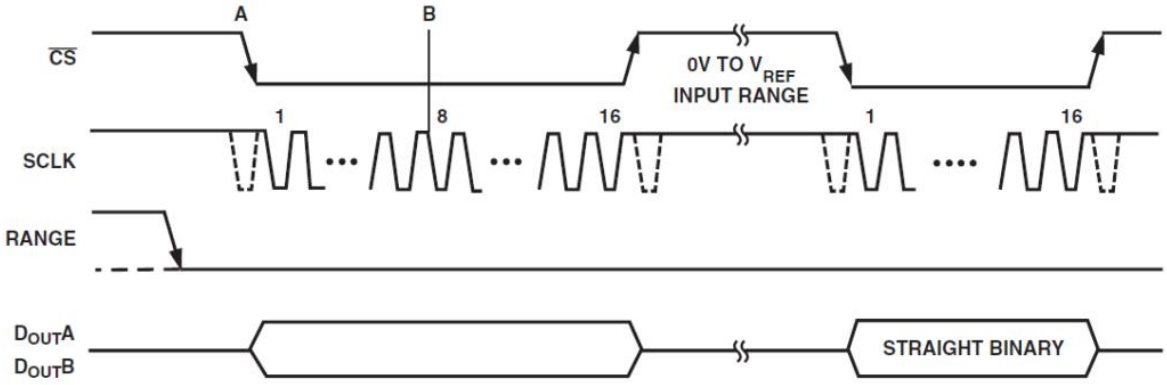
Die von den Strom-Messwandlern gemessenen Phasenstr□öme sollen von der Br□ücke digital an das zur Ansteuerung verwendete Zielsystem □übertragen werden. Das

Zielsystem kann diese Messwerte dann verwenden, um beispielsweise einen Stromregler (z.B. f□ür feldorientierte Vektorregelung), eine Nulldurchgangserkennung (z.B. f□ur geberlose Ansteuerung) oder eine □Überstromabschaltung zu implementieren. Hierf□ür ist es notwendig, min. zwei Phasenstr□öme zeitgleich zu samplen. Der A/D-Wandler AD7866 (Analog Devices) ist f□ür diese Aufgabe ideal geeignet:

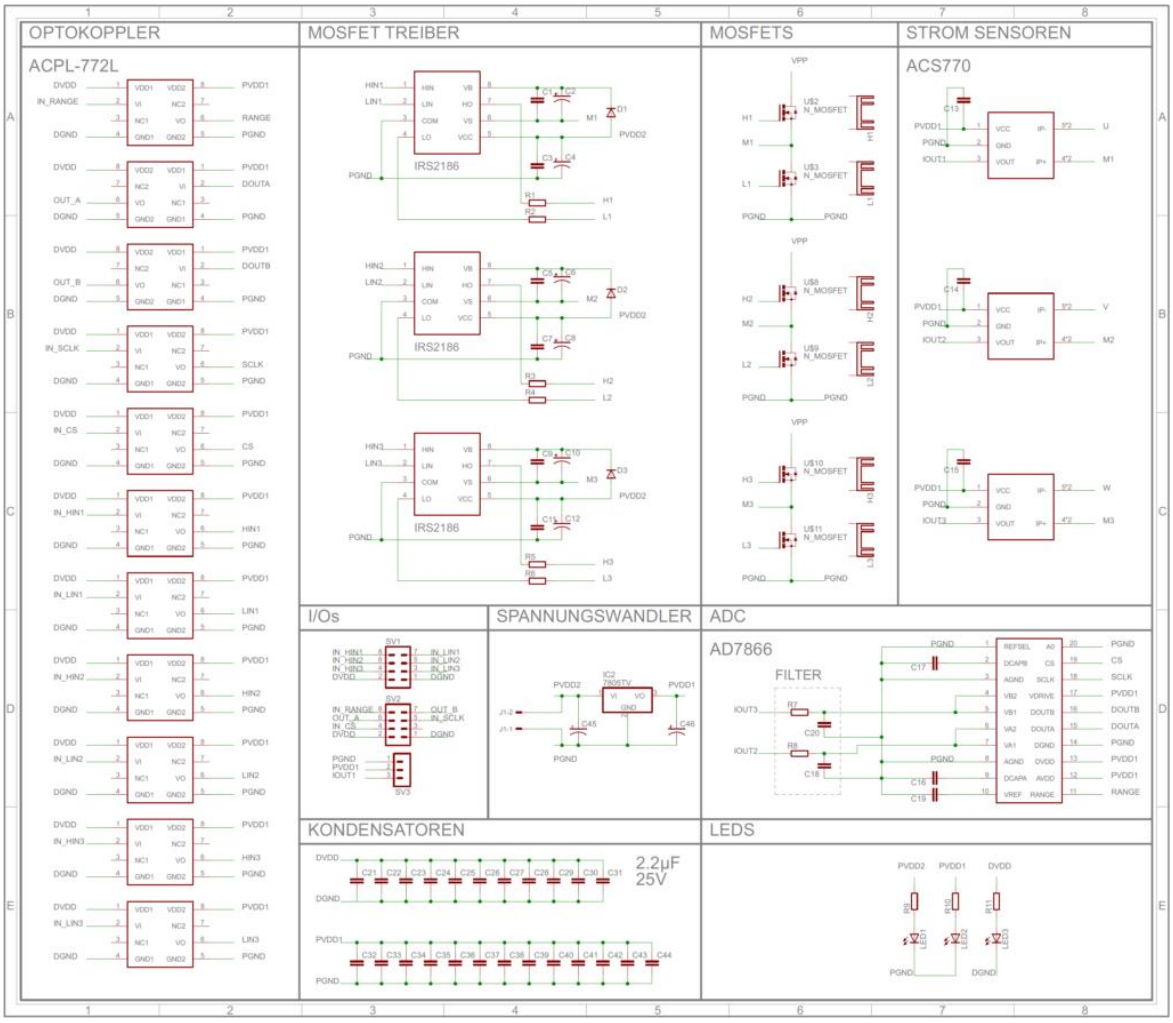


Range Level @ Point A[1]	Range Level @ Point B[2]	Input Range[3]	Output Coding[3]
Low	Low	0 V to V_{REF}	Straight Binary
High	High	$V_{REF} \pm V_{REF}$	Twos Complement
Low	High	$V_{REF}/2 \pm V_{REF}/2$	Twos Complement
High	Low	0 V to 2 × V_{REF}	Straight Binary

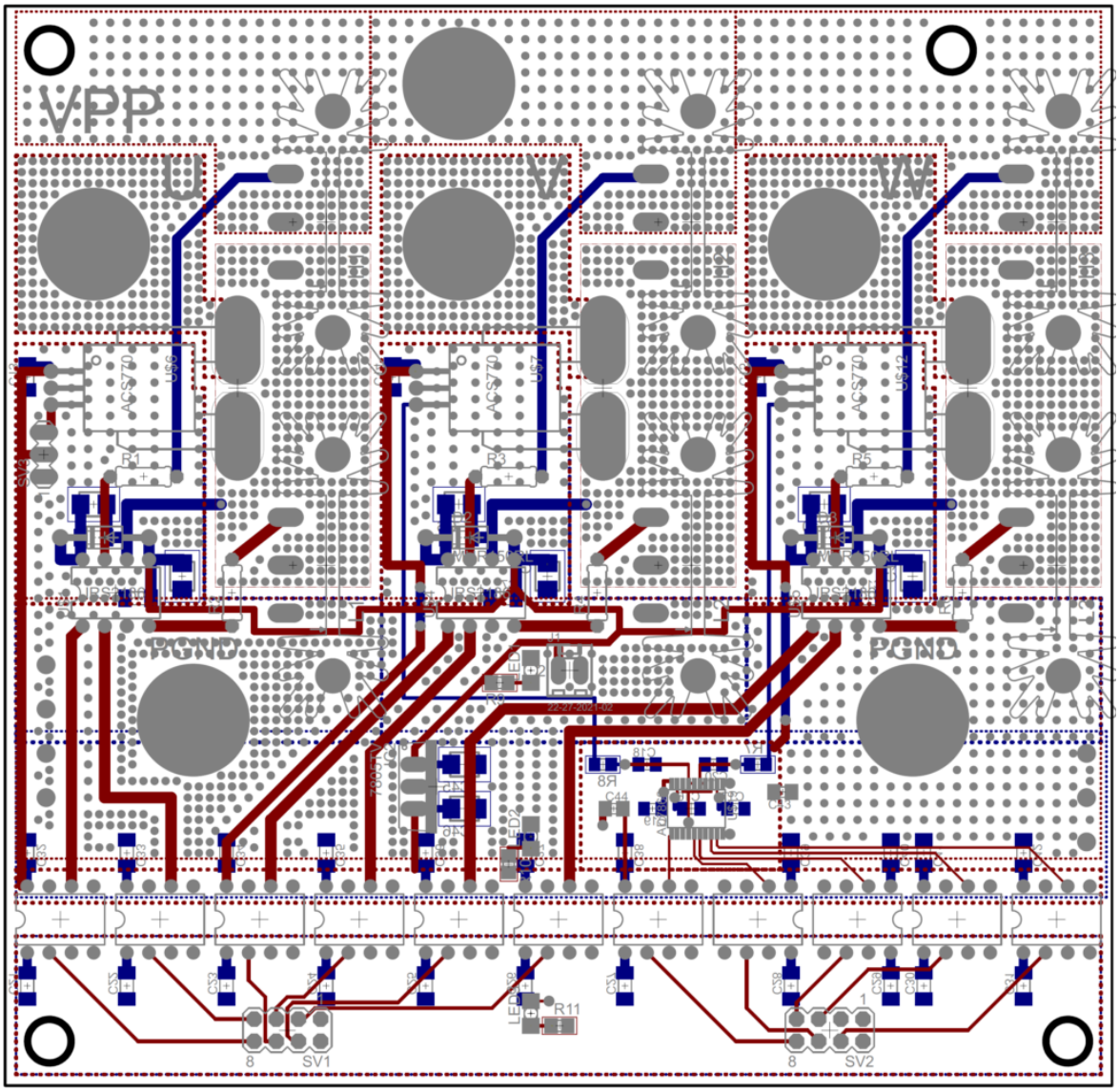
NOTES
[1]Point A = Falling edge of $\overline{CS}$.
[2]Point B = Eighth falling edge of SCLK.
[3]Selected for next conversion.

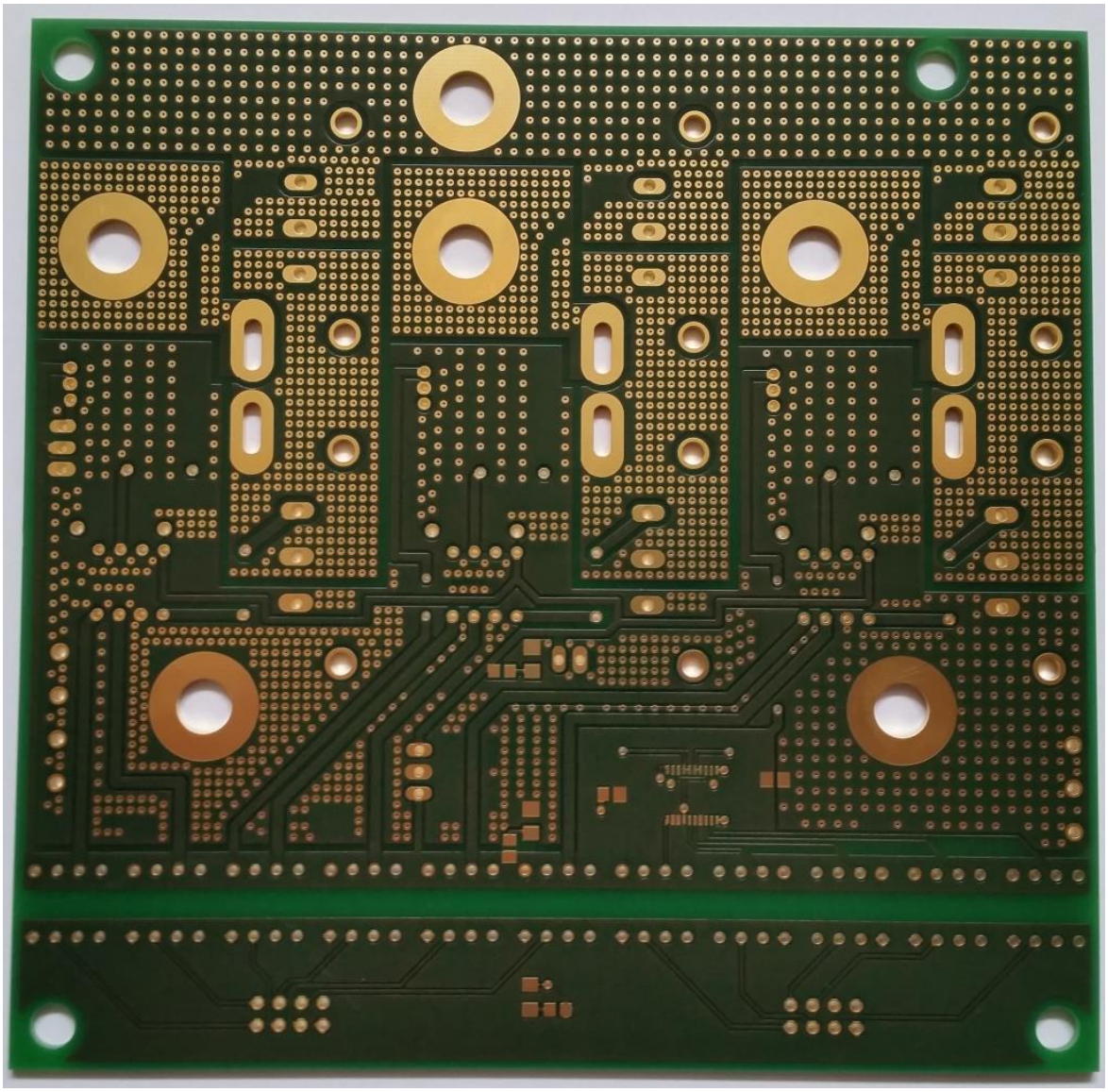


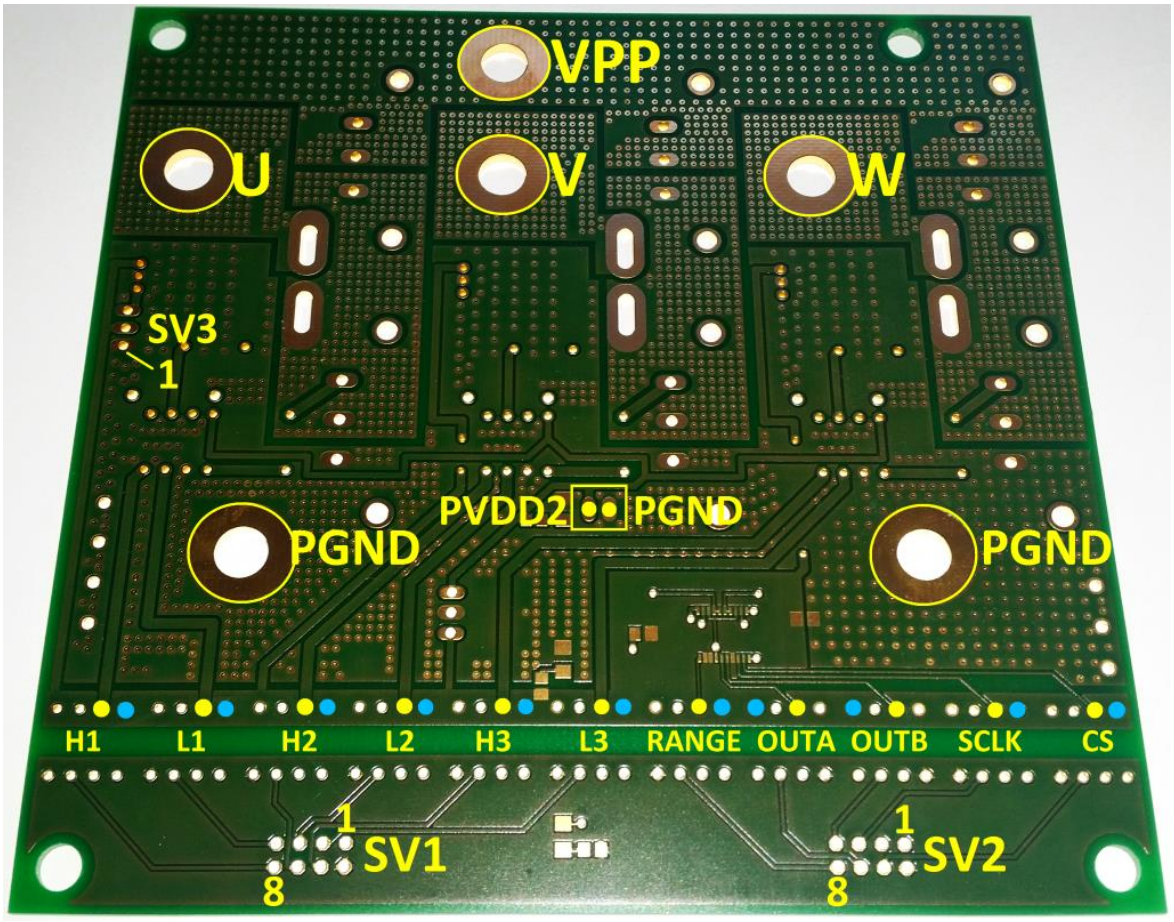
Schaltplan

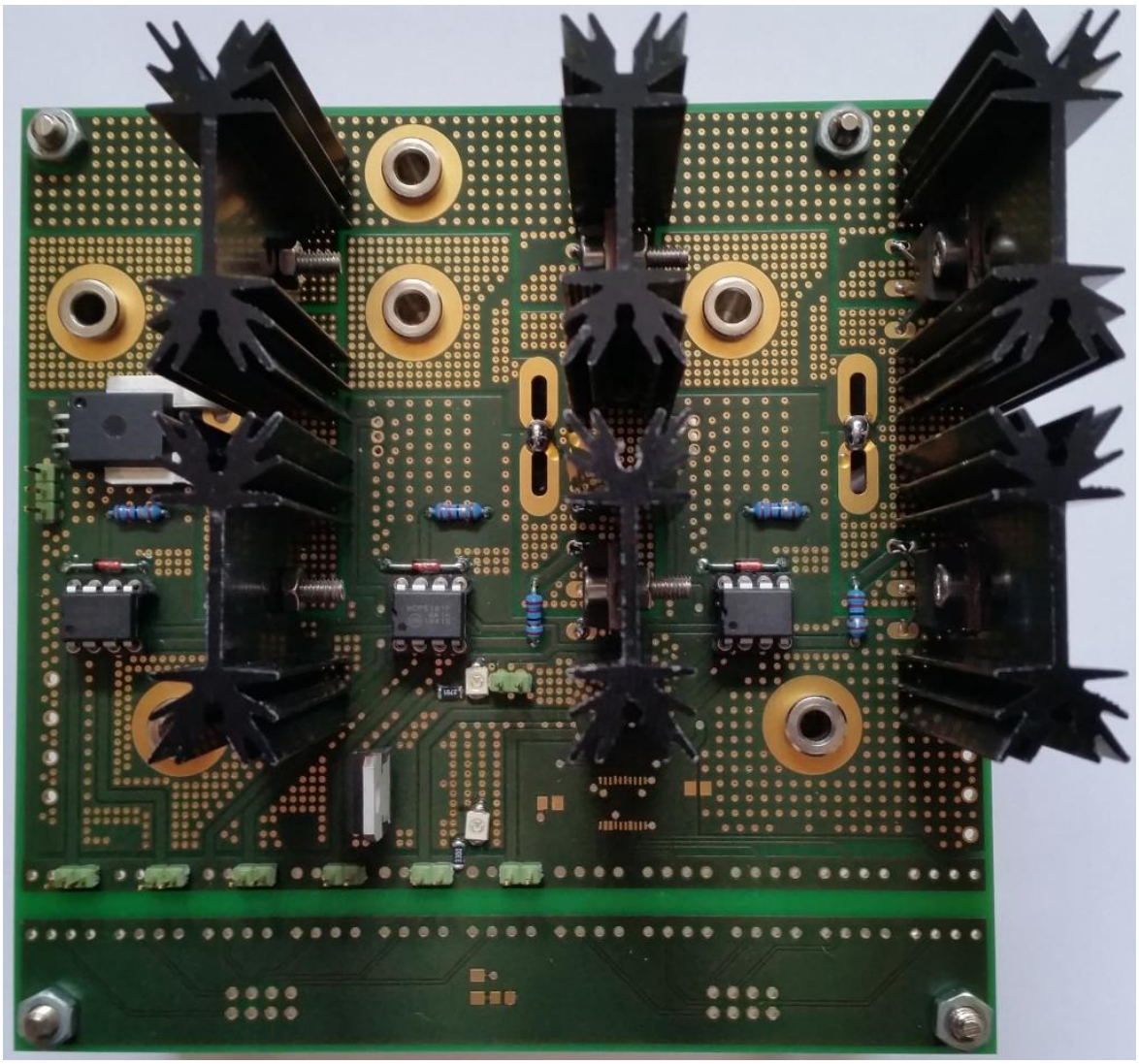

OPTOKOPPLER
ACPL-772L
MOSFET TREIBER
IRS2186
IRS2186
IRS2186
MOSFETS
STROM SENSOREN
ACS770
I/Os
SPANNUNGSWANDLER
ADC
AD7866
FILTER
KONDENSATOREN
2.2µF
25V
LEDS

Layout


VPP
U
V
W
PGND
PGND
ACS770
ACS770
ACS770
SV1
SV2
R11




VPP
U
V
W
SV3
1
PVDD2
PGND
PGND
PGND
H1
L1
H2
L2
H3
L3
RANGE
OUTA
OUTB
SCLK
CS
1
SV1
8
1
SV2
8



Software

```
/* Defines */

#define  F_PWM       10000

#define  MAX_CNT     ((SystemCoreClock / F_PWM) - 1)

#define  PI          3.141592



/* Globales Sinusarray */

uint16_t sin_arr[360];
```

```
/* Funktion initialisiert die 3-Phasen-PWM */

void init_3_phase_pwm( void )

{

   TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;

   TIM_OCInitTypeDef  TIM_OCInitStructure;

   TIM_BDTRInitTypeDef TIM_BDTRInitStructure;

   GPIO_InitTypeDef GPIO_InitStructure;



   RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

   RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

   RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);



   /* PWM-Funktion der Pins aktivieren */

   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9 |
GPIO_Pin_10;

   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;

   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;

   GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;

   GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;

   GPIO_Init(GPIOA, &GPIO_InitStructure);

   GPIO_PinAFConfig(GPIOA, GPIO_PinSource7, GPIO_AF_TIM1);
```

```
GPIO_PinAFConfig(GPIOA, GPIO_PinSource8, GPIO_AF_TIM1);
GPIO_PinAFConfig(GPIOA, GPIO_PinSource9, GPIO_AF_TIM1);
GPIO_PinAFConfig(GPIOA, GPIO_PinSource10, GPIO_AF_TIM1);

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_14 | GPIO_Pin_15;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOB, &GPIO_InitStructure);
GPIO_PinAFConfig(GPIOB, GPIO_PinSource14, GPIO_AF_TIM1);
GPIO_PinAFConfig(GPIOB, GPIO_PinSource15, GPIO_AF_TIM1);

/* Timer konfigurieren */
TIM_TimeBaseStructure.TIM_Prescaler = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseStructure.TIM_Period = MAX_CNT;
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_RepetitionCounter = 0;
TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure);
```

```
/* PWM 1,2 und 3 konfigurieren */

TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM2;

TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;

TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Enable;

TIM_OCInitStructure.TIM_Pulse = 0;

TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_High;

TIM_OCInitStructure.TIM_OCIdleState = TIM_OCIdleState_Set;

TIM_OCInitStructure.TIM_OCNIdleState = TIM_OCIdleState_Reset;

TIM_OC1Init(TIM1, &TIM_OCInitStructure);

TIM_OC2Init(TIM1, &TIM_OCInitStructure);

TIM_OC3Init(TIM1, &TIM_OCInitStructure);



/* Totzeit konfigurieren */

TIM_BDTRInitStructure.TIM_OSSRState = TIM_OSSRState_Enable;

TIM_BDTRInitStructure.TIM_OSSIState = TIM_OSSIState_Enable;

TIM_BDTRInitStructure.TIM_LOCKLevel = TIM_LOCKLevel_1;

TIM_BDTRInitStructure.TIM_DeadTime = 100;

TIM_BDTRInitStructure.TIM_Break = TIM_Break_Disable;

TIM_BDTRInitStructure.TIM_BreakPolarity = TIM_BreakPolarity_High;

TIM_BDTRInitStructure.TIM_AutomaticOutput = TIM_AutomaticOutput_Enable;
```

```
   TIM_BDTRConfig(TIM1, &TIM_BDTRInitStructure);



   /* Timer einschalten */

   TIM_Cmd(TIM1, ENABLE);



   /* PWM-Ausgänge aktivieren */

   TIM_CtrlPWMOutputs(TIM1, ENABLE);

}
```

```
/* Funktion füllt ein Sinusarray */

void init_sin_arr( void )

{

   int n;



   /* Sinusarray im Abstand von einem Grad füllen */

   for(n=0;n<360;n++)

   {

      /* Duty für den jeweiligen Winkel berechnen */

      sin_arr[n] = (uint16_t) ( MAX_CNT * 0.5 * (1.0 + sin( 2 * PI * n / 360)
) + 0.5 );

   }

}
```

```
/* Funktion initialisiert den externen Interrupt */

void init_ext_int(void)

{

    GPIO_InitTypeDef GPIO_InitStruct;

    EXTI_InitTypeDef EXTI_InitStruct;

    NVIC_InitTypeDef NVIC_InitStruct;



    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);



    /* Externen Interrupt konfigurieren */

    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;

    GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;

    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_0;

    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;

    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_100MHz;

    GPIO_Init(GPIOB, &GPIO_InitStruct);

    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource0);



    /* Auf steigende Flanke triggern */

    EXTI_InitStruct.EXTI_Line = EXTI_Line0;
```

```
    EXTI_InitStruct.EXTI_LineCmd = ENABLE;
    EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Rising;
    EXTI_Init(&EXTI_InitStruct);


    /* Interrupt einschalten */
    NVIC_InitStruct.NVIC_IRQChannel = EXTI0_IRQn;
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x00;
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x00;
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStruct);
}


/* ISR des externen Interrupts */
void EXTI0_IRQHandler( void )
{
   static int angle;
   if ( EXTI->PR & EXTI_Line0 )
   {
         /* PWM 1 --> sin(a + 0°)             */
      TIM1->CCR1 = sin_arr[  angle         % 360 ] + 50;
```

```
      /* PWM 2 --> sin(a + 120°)           */
      TIM1->CCR2 = sin_arr[ (angle + 120 ) % 360 ] + 50;
      /* PWM 3 --> sin(a + 240°)           */
      TIM1->CCR3 = sin_arr[ (angle + 240 ) % 360 ] + 50;
      /* Winkel um ein Grad inkrementieren */
      angle++;


      if ( !(angle % 180) ) GPIOA->ODR ^= GPIO_Pin_5;
      if ( !(angle % 360) ) angle = 0;


      EXTI->PR = EXTI_Line0;
   }
}
/* AD9850 */
#define AD9850_RESET       GPIOC, GPIO_Pin_1
#define AD9850_SCK         GPIOC, GPIO_Pin_3
#define AD9850_DATA        GPIOC, GPIO_Pin_2
#define AD9850_LATCH       GPIOC, GPIO_Pin_0


void ad9850_init( void )
{
```

```
   GPIO_WriteBit(AD9850_SCK, Bit_RESET);
   GPIO_WriteBit(AD9850_LATCH, Bit_RESET);
   GPIO_WriteBit(AD9850_RESET, Bit_RESET);
   delay_ms(10);
   GPIO_WriteBit(AD9850_RESET, Bit_SET);
   delay_ms(10);
   GPIO_WriteBit(AD9850_RESET, Bit_RESET);


   GPIO_WriteBit(AD9850_LATCH, Bit_SET);
   delay_ms(10);
   GPIO_WriteBit(AD9850_LATCH, Bit_RESET);
   delay_ms(10);
}


void ad9850_write_byte( unsigned char b )
{
   unsigned char i;
   GPIO_WriteBit(AD9850_SCK, Bit_RESET);
   for (i=0; i<8; i++)
   {
      GPIO_WriteBit(AD9850_DATA, (b&0x01)?Bit_SET:Bit_RESET);
```

```
      b>>=1;
      GPIO_WriteBit(AD9850_SCK, Bit_RESET);
      GPIO_WriteBit(AD9850_SCK, Bit_SET);
   }
}

void ad9850_set_freq( double f )
{
   unsigned long int y;
   f/=1000000;
   f=f*(4294967295/125);
   y=f;

   GPIO_WriteBit(AD9850_LATCH, Bit_RESET);

   ad9850_write_byte( y & 0xFF );
   y>>=8;
   ad9850_write_byte( y & 0xFF );
   y>>=8;
   ad9850_write_byte( y & 0xFF );
   y>>=8;
```

```
   ad9850_write_byte( y & 0xFF );


   ad9850_write_byte( 0 );


   GPIO_WriteBit(AD9850_LATCH, Bit_SET);
   GPIO_WriteBit(AD9850_LATCH, Bit_RESET);
}
int main(void)
{
   SystemInit();                    /* MCU initialisieren               */
   SystemCoreClockUpdate();         /* Kernfrequenz updaten             */
   delay_init();                    /* Delay-Funktionen initialisieren  */
   ports_init();                    /* Ports initialisieren             */
   init_sin_arr();                  /* Sinusarray vorbereiten           */
   init_3_phase_pwm();              /* 3-Phasen-PWM initialisieren      */
   if (SysTick_Config(SystemCoreClock / 1000)) while (1);


   ad9850_init();                   /* AD9850 initialisieren            */
   init_ext_int();                  /* Externen Interrupt konfigurieren */


   ad9850_set_freq( 10 * 360.0 );
```

```
   while(1)

   {

   }

}
```