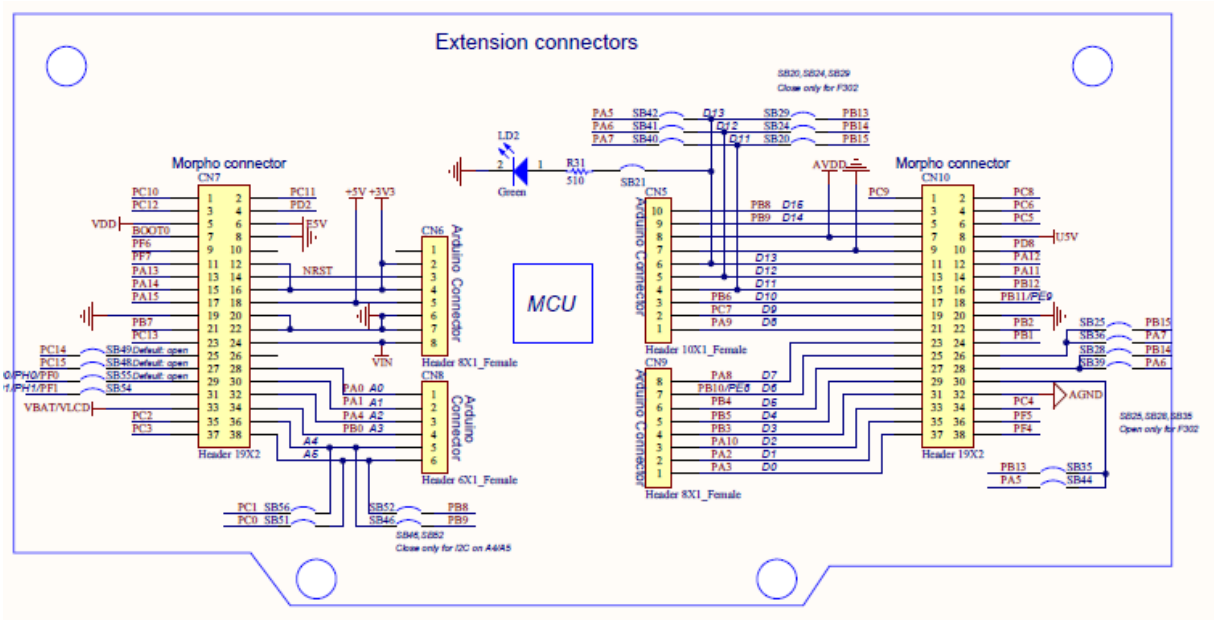
Datenübertragung mit dem SPI-Bus

Verdrahtung Nucleo-Board-SPI-Arduino:

Nucleo	**SPI**	**Arduino**
PA7	MOSI	D11
PA6	MISO	D12
PA5	SCK	D13
?	/CS	D10



Starten Sie die ***OpenSTM32 IDE*** und erstellen Sie ein neues Projekt nach den Vorgaben der ersten Programmieraufgabe. Damit erhalten Sie bereits ein Programmgerüst mit einer leeren Endlosschleife innerhalb eines Hauptprogramms.

!!!! Die Reihenfolge der beschriebenen Codeschnippsel muss eingehalten werden !!!!

1. Systemtakt einstellen

Der erste Funktionsaufruf in main() dient der Takteinstellung. Die Definition der Funktion ***SystemClock_Config()*** ist am Ende der Praktikumsanleitung für den I2C-Bus abgedruckt. Die Einstellung liefert einen Systemtakt für die Peripherie von f = 48 MHz.

```
int main(void)
{

  HAL_Init();
  SystemClock_Config();

  . . . . .
```

2. GPIO- und SPI-Bus Takt einschalten

Für das Einschalten der benötigten Clocks für die Peripherie können Sie die folgenden Makros verwenden, exemplarisch gezeigt für die Ports GPIOA bis GPIOF:

```
/* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOF_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

/* SPI-Bus Clock Enable */
. . . . . .        // wie lautet dieses Makro??
```

3. GPIO-Pins konfigurieren

Sie müssen nun zuerst die Pins für die Verwendung als SCK, MOSI, MISO und /CS einstellen. Dies wird wie auch bei der I2C-Busaufgabe mit einer Variablen des Datentyps *struct.* Die Elemente dieser Variablen müssen Sie lediglich mit den passenden Werten beschreiben und anschließend eine vorgegebene Funktion aufrufen die diese Werte dann in die Register schreibt.

```
   GPIO_InitTypeDef G;                                                    //1)

    /**SPI1 GPIO Configuration
    PA5     ------> SPI1_SCK
    PA6     ------> SPI1_MISO
    PA7     ------> SPI1_MOSI
    */
    G.Pin = GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7;
    G.Mode = GPIO_MODE_AF_PP;
    G.Pull = GPIO_NOPULL;
    G.Speed = GPIO_SPEED_FREQ_HIGH;
    G.Alternate = GPIO_AF0_SPI1 oder GPIO_AF1_SPI1 . . .etc ;          //2)

    . . . . . .                                                        //3)
```

Den oben gezeigten Codeschnippsel kopieren Sie in ihre main() Funktion. Die einzelnen Zeilen sind nachfolgend erläutert:

1)
Zuerst müssen Sie eine Variable deklarieren mit dem passenden Datentyp, für die GPIO ist dies die Struktur **GPIO_InitTypeDef**. Sie finden diese struct in dem folgenden Header **stm32f0xx_hal_gpio.h:**

```
/* @brief   GPIO Init structure definition  */
typedef struct
{
  uint32_t Pin;       /*!< Specifies the GPIO pins to be configured.
                           This parameter can be any value of @ref GPIO_pins */

  uint32_t Mode;      /*!< Specifies the operating mode for the selected pins.
                           This parameter can be a value of @ref GPIO_mode */

  uint32_t Pull;      /*!< Pull-up or Pull-Down activation for the selected pins.
                           This parameter can be a value of @ref GPIO_pull */
```

```
  uint32_t Speed;      /*!< Specifies the speed for the selected pins.
                            This parameter can be a value of @ref GPIO_speed */

  uint32_t Alternate;  /*!< Peripheral to be connected to the selected pins
                             This parameter can be a value of @ref
                              GPIOEx_Alternate_function_selection */
}GPIO_InitTypeDef;
```

Die Bezeichner für GPIO_pins, GPIO_mode , GPIO_pull und GPIO_speed finden Sie in dem Headerfile **stm32f0xx_hal_gpio.h.**

Die Bezeichner für GPIOx_Alternate_ function_selection finden Sie in dem Headerfile **stm32f0xx_hal_gpio_ex.h.**

2)
Welche Alternativ-funktion müssen Sie für den SPI1 Kanal einstellen?

(Tabelle aus dem Datenbuch stm32f072).

3)
Welche Funktion müssen Sie hier noch aufrufen?

Table 14. Alternate functions selected through GPIOA_AFR registers for port A

Pin name	AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7
PA0	-	USART2_CTS	TIM2_CH1_ETR	TSC_G1_IO1	USART4_TX	-	-	COMP1_OUT
PA1	EVENTOUT	USART2_RTS	TIM2_CH2	TSC_G1_IO2	USART4_RX	TIM15_CH1N	-	-
PA2	TIM15_CH1	USART2_TX	TIM2_CH3	TSC_G1_IO3	-	-	-	COMP2_OUT
PA3	TIM15_CH2	USART2_RX	TIM2_CH4	TSC_G1_IO4	-	-	-	-
PA4	SPI1_NSS, I2S1_WS	USART2_CK	-	TSC_G2_IO1	TIM14_CH1	-	-	-
PA5	SPI1_SCK, I2S1_CK	CEC	TIM2_CH1_ETR	TSC_G2_IO2	-	-	-	-
PA6	SPI1_MISO, I2S1_MCK	TIM3_CH1	TIM1_BKIN	TSC_G2_IO3	USART3_CTS	TIM16_CH1	EVENTOUT	COMP1_OUT
PA7	SPI1_MOSI, I2S1_SD	TIM3_CH2	TIM1_CH1N	TSC_G2_IO4	TIM14_CH1	TIM17_CH1	EVENTOUT	COMP2_OUT
PA8	MCO	USART1_CK	TIM1_CH1	EVENTOUT	CRS_SYNC	-	-	-
PA9	TIM15_BKIN	USART1_TX	TIM1_CH2	TSC_G4_IO1	-	-	-	-
PA10	TIM17_BKIN	USART1_RX	TIM1_CH3	TSC_G4_IO2	-	-	-	-
PA11	EVENTOUT	USART1_CTS	TIM1_CH4	TSC_G4_IO3	CAN_RX	-	-	COMP1_OUT
PA12	EVENTOUT	USART1_RTS	TIM1_ETR	TSC_G4_IO4	CAN_TX	-	-	COMP2_OUT
PA13	SWDIO	IR_OUT	USB_NOE	-	-	-	-	-
PA14	SWCLK	USART2_TX	-	-	-	-	-	-
PA15	SPI1_NSS, I2S1_WS	USART2_RX	TIM2_CH1_ETR	EVENTOUT	USART4_RTS	-	-	-

Den Pin für das /CS müssen Sie softwaremäßig bedienen. Welcher Pin ist dies?

```
  GPIO_InitTypeDef G;

   /**SPI1 GPIO Configuration für /CS
   */
   G.Pin = GPIO_PIN_irgendwas;
   G.Mode =  . . . . . ;                          //1)
   G.Pull = GPIO_NOPULL;
   G.Speed = GPIO_SPEED_FREQ_HIGH;

   . . . . . .                                    //2)
```

1)
Für den GPIO-Pin Mode stehen in dem Header **stm32f0xx_hal_gpio.h** unter anderem die folgenden Bezeichner als Präprozessordefines zur Auswahl:

```
GPIO_MODE_INPUT      /*!< Input Floating Mode                   */
```

```
GPIO_MODE_OUTPUT_PP /*!< Output Push Pull Mode                 */
GPIO_MODE_OUTPUT_OD /*!< Output Open Drain Mode                */
GPIO_MODE_AF_PP     /*!< Alternate Function Push Pull Mode     */
GPIO_MODE_AF_OD     /*!< Alternate Function Open Drain Mode    */
GPIO_MODE_ANALOG    /*!< Analog Mode  */
```

2)
Welche Funktion müssen Sie hier noch aufrufen?

Damit sind die nötigen Pins für die Verwendung als SPI-Bus Signale eingestellt und Sie können den SPI-Bus konfigurieren

4. SPI-Bus konfigurieren

```
???????  hspi1;    /*welchen Datentyp müssen Sie hier einsetzen? */

hspi1.Instance = SPI1;
hspi1.Init.Mode = SPI_MODE_MASTER;
hspi1.Init.Direction = SPI_DIRECTION_2LINES;
hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
hspi1.Init.NSS = SPI_NSS_SOFT;
hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_4;         //1)
hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
hspi1.Init.CRCPolynomial = 7;
hspi1.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
hspi1.Init.NSSPMode = SPI_NSS_PULSE_ENABLE;
```

1) Hier wird die Taktrate des SPI Bus eingestellt. In dem Beispiel ist f_SPI = 48MHz/4 = 12 MHz eingestellt.

Das Senden und empfangen der Daten ist nun einfach:

```
uint8_t  TxBuf[] ="Hallo";
uint8_t  RxBuf[5];

HAL_SPI_TransmitReceive(&hspi1, TxBuf, RxBuf, 5, timeout);
```

Das Ergebnis des Lesevorgangs wird in das Feld RxBuf geschrieben.