STM32 HAL-Funktionen

Eine Auswahl von HAL-Funktionen auf einen Blick

1.0

GPIO

Polling

`GPIO_PinState` **`HAL_GPIO_ReadPin`** `(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)`	GPIO-Pin lesen
`void` **`HAL_GPIO_WritePin`** `(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)`	GPIO-Pin schreiben
`void` **`HAL_GPIO_TogglePin`** `(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)`	GPIO-Pin invertieren

Definitionen

`GPIO_PinState``: GPIO_PIN_RESET, GPIO_PIN_SET`	Auswahl logischer Zustand des GPIO-Pins
`GPIO_TypeDef``: GPIOA, GPIOB, GPIOC, ...`	GPIO Modul
`GPIO_Pin``: GPIO_PIN_0, GPIO_PIN_1, GPIO_PIN_2, ...`	GPIO Bitmaske

EXTI

Callback-Funktionen

`void` **`HAL_GPIO_EXTI_Falling_Callback`** `(uint16_t GPIO_Pin)`	INT-Callback für steigende Flanke an GPIO-Pin
`void` **`HAL_GPIO_EXTI_Rising_Callback`** `(uint16_t GPIO_Pin)`	INT-Callback für fallende Flanke an GPIO-Pin

Definitionen

`GPIO_Pin``: GPIO_PIN_0, GPIO_PIN_1, GPIO_PIN_2, ...`	GPIO Bitmaske

Systick

Callback-Funktionen

`void` **`HAL_SYSTICK_Callback`** `(void)`	Systick INT-Callback *(muss in Systick_Handler eingetragen werden, vgl. Abschn. 14.8.3)*

DMA

Polling

`HAL_StatusTypeDef` **`HAL_DMA_Start`** `(DMA_HandleTypeDef* hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength);`	DMA-Transfer starten
`HAL_StatusTypeDef` **`HAL_DMA_PollForTransfer`** `(DMA_HandleTypeDef* hdma, HAL_DMA_LevelCompleteTypeDef CompleteLevel, uint32_t Timeout);`	Mit Polling auf DMA-Ereignis (half complete bzw. complete, s.u.) warten
`HAL_StatusTypeDef` **`HAL_DMA_Abort`** `(DMA_HandleTypeDef* hdma);`	DMA-Transfer abbrechen

Interrupt

`HAL_StatusTypeDef` **`HAL_DMA_Start_IT`** `(DMA_HandleTypeDef* hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength);`	DMA-Transfer starten (mit INT)
`HAL_StatusTypeDef` **`HAL_DMA_Abort_IT`** `(DMA_HandleTypeDef* hdma);`	DMA-Transfer abbrechen (mit INT)

Definitionen

`HAL_DMA_LevelCompleteTypeDef``: HAL_DMA_FULL_TRANSFER, HAL_DMA_HALF_TRANSFER` *(Auswahl des DMA-Ereignisses)*

UART	
Polling	
`HAL_StatusTypeDef` **`HAL_UART_Transmit`** `(UART_HandleTypeDef* huart, uint8_t* pData, uint16_t Size, uint32_t Timeout);`	UART: Senden mit Polling
`HAL_StatusTypeDef` **`HAL_UART_Receive`** `(UART_HandleTypeDef* huart, uint8_t* pData, uint16_t Size, uint32_t Timeout);`	UART: Empfangen mit Polling
Interrupt	
`HAL_StatusTypeDef` **`HAL_UART_Transmit_IT`** `(UART_HandleTypeDef* huart, uint8_t* pData, uint16_t Size);`	UART: Senden mit INT
`HAL_StatusTypeDef` **`HAL_UART_Receive_IT`** `(UART_HandleTypeDef* huart, uint8_t* pData, uint16_t Size);`	UART: Empfangen mit INT
DMA	
`HAL_StatusTypeDef` **`HAL_UART_Transmit_DMA`** `(UART_HandleTypeDef* huart, uint8_t* pData, uint16_t Size);`	UART: Senden mit DMA
`HAL_StatusTypeDef` **`HAL_UART_Receive_DMA`** `(UART_HandleTypeDef* huart, uint8_t* pData, uint16_t Size);`	UART: Empfangen mit DMA
`HAL_StatusTypeDef` **`HAL_UART_DMAPause`** `(UART_HandleTypeDef* huart);`	UART: DMA-Transfers pausieren
`HAL_StatusTypeDef` **`HAL_UART_DMAResume`** `(UART_HandleTypeDef* huart);`	UART: DMA-Transfers fortsetzen
`HAL_StatusTypeDef` **`HAL_UART_DMAStop`** `(UART_HandleTypeDef* huart);`	UART: DMA-Transfers stoppen
`HAL_StatusTypeDef` **`HAL_UART_Abort`** `(UART_HandleTypeDef* huart);`	UART: DMA-Transfers abbrechen
Callback-Funktionen	
`void` **`HAL_UART_TxCpltCallback`** `(UART_HandleTypeDef* huart);`	UART-CB: Senden abgeschlossen
`void` **`HAL_UART_RxCpltCallback`** `(UART_HandleTypeDef* huart);`	UART-CB: Empfangen abgeschlossen
`void` **`HAL_UART_TxHalfCpltCallback`** `(UART_HandleTypeDef* huart);`	UART-CB: Senden mit DMA halb abgeschlossen
`void` **`HAL_UART_TxCpltCallback`** `(UART_HandleTypeDef* huart);`	UART-CB: Senden mit DMA abgeschlossen
`void` **`HAL_UART_RxHalfCpltCallback`** `(UART_HandleTypeDef* huart);`	UART-CB: Empfangen mit DMA halb abgeschlossen
`void` **`HAL_UART_RxCpltCallback`** `(UART_HandleTypeDef* huart);`	UART-CB: Empfangen mit DMA abgeschlossen
`void` **`HAL_UART_ErrorCallback`** `(UART_HandleTypeDef* huart);`	UART-CB: Fehler (z.B. beim Datentransfer) aufgetreten

SPI	
Polling	
`HAL_StatusTypeDef` **`HAL_SPI_Transmit`** `(SPI_HandleTypeDef* hspi, uint8_t* pData, uint16_t Size, uint32_t Timeout);`	SPI: Senden mit Polling
`HAL_StatusTypeDef` **`HAL_SPI_Receive`** `(SPI_HandleTypeDef* hspi, uint8_t* pData, uint16_t Size, uint32_t Timeout);`	SPI: Empfangen mit Polling
`HAL_StatusTypeDef` **`HAL_SPI_TransmitReceive`** `(SPI_HandleTypeDef* hspi, uint8_t* pTxData, uint8_t* pRxData, uint16_t Size, uint32_t Timeout);`	SPI: Senden & Empfangen mit Polling
Interrupt	
`HAL_StatusTypeDef` **`HAL_SPI_Transmit_IT`** `(SPI_HandleTypeDef* hspi, uint8_t* pData, uint16_t Size);`	SPI: Senden mit INT
`HAL_StatusTypeDef` **`HAL_SPI_Receive_IT`** `(SPI_HandleTypeDef* hspi, uint8_t* pData, uint16_t Size);`	SPI: Empfangen mit INT
`HAL_StatusTypeDef` **`HAL_SPI_TransmitReceive_IT`** `(SPI_HandleTypeDef* hspi, uint8_t* pTxData, uint8_t* pRxData, uint16_t Size);`	SPI: Senden & Empfangen mit INT
DMA	
`HAL_StatusTypeDef` **`HAL_SPI_Transmit_DMA`** `(SPI_HandleTypeDef* hspi, uint8_t* pData, uint16_t Size);`	SPI: Senden mit DMA
`HAL_StatusTypeDef` **`HAL_SPI_Receive_DMA`** `(SPI_HandleTypeDef* hspi, uint8_t* pData, uint16_t Size);`	SPI: Empfangen mit DMA
`HAL_StatusTypeDef` **`HAL_SPI_TransmitReceive_DMA`** `(SPI_HandleTypeDef* hspi, uint8_t* pTxData, uint8_t* pRxData, uint16_t Size);`	SPI: Senden & Empfangen mit DMA
`HAL_StatusTypeDef` **`HAL_SPI_DMAPause`** `(SPI_HandleTypeDef* hspi);`	SPI: DMA-Transfers pausieren
`HAL_StatusTypeDef` **`HAL_SPI_DMAResume`** `(SPI_HandleTypeDef* hspi);`	SPI: DMA-Transfers fortsetzen
`HAL_StatusTypeDef` **`HAL_SPI_DMAStop`** `(SPI_HandleTypeDef* hspi);`	SPI: DMA-Transfers stoppen
`HAL_StatusTypeDef` **`HAL_SPI_Abort`** `(SPI_HandleTypeDef* hspi);`	SPI: DMA-Transfers abbrechen
Callback-Funktionen	
`void` **`HAL_SPI_TxCpltCallback`** `(SPI_HandleTypeDef* hspi);`	SPI-CB: Senden abgeschlossen
`void` **`HAL_SPI_RxCpltCallback`** `(SPI_HandleTypeDef* hspi);`	SPI-CB: Empfangen abgeschlossen
`void` **`HAL_SPI_TxRxCpltCallback`** `(SPI_HandleTypeDef* hspi);`	SPI-CB: Senden & Empfangen abgeschlossen
`void` **`HAL_SPI_TxHalfCpltCallback`** `(SPI_HandleTypeDef* hspi);`	SPI-CB: Senden mit DMA halb abgeschlossen
`void` **`HAL_SPI_TxCpltCallback`** `(SPI_HandleTypeDef* hspi);`	SPI-CB: Senden mit DMA abgeschlossen
`void` **`HAL_SPI_RxHalfCpltCallback`** `(SPI_HandleTypeDef* hspi);`	SPI-CB: Empfangen mit DMA halb abgeschlossen
`void` **`HAL_SPI_RxCpltCallback`** `(SPI_HandleTypeDef* hspi);`	SPI-CB: Empfangen mit DMA abgeschlossen
`void` **`HAL_SPI_ErrorCallback`** `(SPI_HandleTypeDef* hspi);`	SPI-CB: Fehler aufgetreten

I²C

Polling	
`HAL_StatusTypeDef` **`HAL_I2C_Master_Transmit`** `(I2C_HandleTypeDef* hi2c, uint16_t DevAddress, uint8_t* pData, uint16_t Size, uint32_t Timeout);`	I2C: Senden mit Polling
`HAL_StatusTypeDef` **`HAL_I2C_Master_Receive`** `(I2C_HandleTypeDef* hi2c, uint16_t DevAddress, uint8_t* pData, uint16_t Size, uint32_t Timeout);`	I2C: Empfangen mit Polling
`HAL_StatusTypeDef` **`HAL_I2C_Mem_Write`** `(I2C_HandleTypeDef* hi2c, uint16_t DevAddress, uint16_t MemAddr, uint8_t MemAddrSize, uint8_t* pData, uint16_t Size, uint32_t Timeout);`	I2C: Senden an registerbasierten Slave mit Polling
`HAL_StatusTypeDef` **`HAL_I2C_Mem_Read`** `(I2C_HandleTypeDef* hi2c, uint16_t DevAddress, uint16_t MemAddr, uint8_t MemAddrSize, uint8_t* pData, uint16_t Size, uint32_t Timeout);`	I2C: Empfangen von registerbasierten Slave mit Polling
Interrupt	
`HAL_StatusTypeDef` **`HAL_I2C_Master_Transmit_IT`** `(I2C_HandleTypeDef* hi2c, uint16_t DevAddress, uint8_t* pData, uint16_t Size);`	I2C: Senden mit INT
`HAL_StatusTypeDef` **`HAL_I2C_Master_Receive_IT`** `(I2C_HandleTypeDef* hi2c, uint16_t DevAddress, uint8_t* pData, uint16_t Size);`	I2C: Empfangen mit INT
`HAL_StatusTypeDef` **`HAL_I2C_Mem_Write_IT`** `(I2C_HandleTypeDef* hi2c, uint16_t DevAddress, uint16_t MemAddr, uint8_t MemAddrSize, uint8_t* pData, uint16_t Size);`	I2C: Senden an registerbasierten Slave mit INT
`HAL_StatusTypeDef` **`HAL_I2C_Mem_Read_IT`** `(I2C_HandleTypeDef* hi2c, uint16_t DevAddress, uint16_t MemAddr, uint8_t MemAddrSize, uint8_t* pData, uint16_t Size);`	I2C: Empfangen von registerbasierten Slave mit INT
DMA	
`HAL_StatusTypeDef` **`HAL_I2C_Master_Transmit_DMA`** `(I2C_HandleTypeDef* hi2c, uint16_t DevAddress, uint8_t* pData, uint16_t Size);`	I2C: Senden mit DMA
`HAL_StatusTypeDef` **`HAL_I2C_Master_Receive_DMA`** `(I2C_HandleTypeDef* hi2c, uint16_t DevAddress, uint8_t* pData, uint16_t Size);`	I2C: Empfangen mit DMA
`HAL_StatusTypeDef` **`HAL_I2C_Mem_Write_DMA`** `(I2C_HandleTypeDef* hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t* pData, uint16_t Size);`	I2C: Senden an registerbasierten Slave mit DMA
`HAL_StatusTypeDef` **`HAL_I2C_Mem_Read_DMA`** `(I2C_HandleTypeDef* hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t* pData, uint16_t Size);`	I2C: Empfangen von registerbasierten Slave mit DMA
Definitionen	
`DevAddress`: "Byte-Addresse" = die 7-Bit-Bausteinadresse befindet sich in den Bits [7:1]	
Callback-Funktionen	
`void` **`HAL_I2C_MasterTxCpltCallback`** `(I2C_HandleTypeDef* hi2c);`	I2C-CB: Senden abgeschlossen
`void` **`HAL_I2C_MasterRxCpltCallback`** `(I2C_HandleTypeDef* hi2c);`	I2C-CB: Empfangen abgeschlossen
`void` **`HAL_I2C_MemTxCpltCallback`**`(I2C_HandleTypeDef* hi2c);`	I2C-CB: Senden abgeschlossen
`void` **`HAL_I2C_MemRxCpltCallback`**`(I2C_HandleTypeDef* hi2c);`	I2C-CB: Empfangen abgeschlossen
`void` **`HAL_I2C_ErrorCallback`**`(I2C_HandleTypeDef* hi2c);`	I2C-CB: Fehler aufgetreten

Timer

Polling	
`HAL_StatusTypeDef` **`HAL_TIM_Base_Start`**`(TIM_HandleTypeDef* htim);`	Timer starten
`HAL_StatusTypeDef` **`HAL_TIM_OC_Start`**`(TIM_HandleTypeDef* htim, uint32_t Channel);`	Timer mit Output-Compare starten
`HAL_StatusTypeDef` **`HAL_TIM_PWM_Start`**`(TIM_HandleTypeDef* htim, uint32_t Channel);`	Timer mit PWM starten
`HAL_StatusTypeDef` **`HAL_TIM_IC_Start`**`(TIM_HandleTypeDef* htim, uint32_t Channel);`	Timer mit Input-Capture starten
`HAL_StatusTypeDef` **`HAL_TIM_Encoder_Start`**`(TIM_HandleTypeDef* htim, uint32_t Channel);`	Timer mit Encoder-Mode starten
`uint32_t` **`HAL_TIM_ReadCapturedValue`**`(TIM_HandleTypeDef* htim, uint32_t Channel);`	Input-Capture-Wert lesen
Interrupt	
`HAL_StatusTypeDef` **`HAL_TIM_Base_Start_IT`**`(TIM_HandleTypeDef* htim);`	Timer starten
`HAL_StatusTypeDef` **`HAL_TIM_OC_Start_IT`**`(TIM_HandleTypeDef* htim, uint32_t Channel);`	Timer mit Output-Compare starten
`HAL_StatusTypeDef` **`HAL_TIM_PWM_Start_IT`**`(TIM_HandleTypeDef* htim, uint32_t Channel);`	Timer mit PWM starten
`HAL_StatusTypeDef` **`HAL_TIM_IC_Start_IT`**`(TIM_HandleTypeDef* htim, uint32_t Channel);`	Timer mit Input-Capture starten
`HAL_StatusTypeDef` **`HAL_TIM_Encoder_Start_IT`**`(TIM_HandleTypeDef* htim, uint32_t Channel);`	Timer mit Encoder-Mode starten
DMA	
`HAL_StatusTypeDef` **`HAL_TIM_Base_Start_DMA`**`(TIM_HandleTypeDef* htim, uint32_t* pData, uint16_t Length);`	Timer starten
`HAL_StatusTypeDef` **`HAL_TIM_OC_Start_DMA`**`(TIM_HandleTypeDef* htim, uint32_t Channel, uint32_t* pData, uint16_t Length);`	Timer mit Output-Compare starten
`HAL_StatusTypeDef` **`HAL_TIM_PWM_Start_DMA`**`(TIM_HandleTypeDef* htim, uint32_t Channel, uint32_t* pData, uint16_t Length);`	Timer mit PWM starten
`HAL_StatusTypeDef` **`HAL_TIM_IC_Start_DMA`**`(TIM_HandleTypeDef* htim, uint32_t Channel, uint32_t* pData, uint16_t Length);`	Timer mit Input-Capture starten
`HAL_StatusTypeDef` **`HAL_TIM_Encoder_Start_DMA`**`(TIM_HandleTypeDef* htim, uint32_t Channel, uint32_t* pData1, uint32_t* pData2, uint16_t Length);`	Timer mit Encoder-Mode starten
`HAL_StatusTypeDef` **`HAL_TIM_DMABurst_WriteStart`**`(TIM_HandleTypeDef* htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t* BurstBuffer, uint32_t BurstLength);`	DMA-Schreib-Transfer in Timer-Register starten
`HAL_StatusTypeDef` **`HAL_TIM_DMABurst_MultiWriteStart`**`(TIM_HandleTypeDef*htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t*BurstBuffer, uint32_t BurstLength, uint32_t DataLength);`	DMA-Schreib-Transfer in mehrere Timer-Register starten
`HAL_StatusTypeDef` **`HAL_TIM_DMABurst_ReadStart`**`(TIM_HandleTypeDef* htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t* BurstBuffer, uint32_t BurstLength);`	DMA-Lese-Transfer in Timer-Register starten
`HAL_StatusTypeDef` **`HAL_TIM_DMABurst_MultiReadStart`**`(TIM_HandleTypeDef*htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t*BurstBuffer, uint32_t BurstLength, uint32_t DataLength);`	DMA-Lese-Transfer in mehrere Timer-Register starten
Definitionen	
`Channel:` `TIM_CHANNEL_1, TIM_CHANNEL_2, ...`	
Callback-Funktionen	

`void` **`HAL_TIM_PeriodElapsedCallback`**`(TIM_HandleTypeDef* htim);`	Zählperiode abgeschlossen
`void` **`HAL_TIM_OC_DelayElapsedCallback`**`(TIM_HandleTypeDef* htim);`	Output-Compare-Ereignis aufgetreten
`void` **`HAL_TIM_PWM_PulseFinishedCallback`**`(TIM_HandleTypeDef* htim);`	PWM Puls ausgegeben
`void` **`HAL_TIM_IC_CaptureCallback`**`(TIM_HandleTypeDef* htim);`	Input-Capture-Ereignis aufgetreten
`void` **`HAL_TIM_TriggerCallback`**`(TIM_HandleTypeDef* htim);`	Trigger-Ereignis aufgetreten
`void` **`HAL_TIM_ErrorCallback`**`(TIM_HandleTypeDef* htim);`	Fehler aufgetreten

Watchdog

Polling	
`HAL_StatusTypeDef` **`HAL_IWDG_Refresh`**`(IWDG_HandleTypeDef* hiwdg);`	Independent Watchdog Refresh
`HAL_StatusTypeDef` **`HAL_WWDG_Refresh`**`(WWDG_HandleTypeDef* hwwdg);`	Window Watchdog Refresh
Callback-Funktionen	
`void` **`HAL_WWDG_EarlyWakeupCallback`**`(WWDG_HandleTypeDef* hwwdg);`	Window Watchdog Early Wakeup

ADC

Polling	
`HAL_StatusTypeDef` **`HAL_ADC_Start`**`(ADC_HandleTypeDef* hadc);`	ADC starten
`HAL_StatusTypeDef` **`HAL_ADC_PollForConversion`**`(ADC_HandleTypeDef* hadc, uint32_t Timeout);`	Auf ADC-Umwandlung warten
`HAL_StatusTypeDef` **`HAL_ADC_PollForEvent`**`(ADC_HandleTypeDef* hadc, uint32_t EventType, uint32_t Timeout);`	Auf ADC-Event warten
`uint32_t` **`HAL_ADC_GetValue`**`(ADC_HandleTypeDef* hadc);`	ADC-Wert lesen
Interrupt	
`HAL_StatusTypeDef` **`HAL_ADC_Start_IT`**`(ADC_HandleTypeDef* hadc);`	ADC starten, mit INT
DMA	
`HAL_StatusTypeDef` **`HAL_ADC_Start_DMA`**`(ADC_HandleTypeDef* hadc, uint32_t* pData, uint32_t Length);`	ADC starten, mit DMA
Definitionen	
`EventType``: ADC_EOSMP_EVENT, ADC_AWD1_EVENT ADC, ADC_AWD2_EVENT, ADC_AWD3_EVENT, ADC_OVR_EVENT`	
Callback-Funktionen	
`void` **`HAL_ADC_ConvCpltCallback`**`(ADC_HandleTypeDef* hadc);`	Conversion complete
`void` **`HAL_ADC_ConvHalfCpltCallback`**`(ADC_HandleTypeDef* hadc);`	Conversion half complete
`void` **`HAL_ADC_ErrorCallback`**`(ADC_HandleTypeDef* hadc);`	Fehler aufgertreten

COMP

Polling	
`HAL_StatusTypeDef` **`HAL_COMP_Start`**`(COMP_HandleTypeDef* hcomp);`	Komparator starten
`uint32_t` **`HAL_COMP_GetOutputLevel`**`(COMP_HandleTypeDef* hcomp);`	Ausgabewert von Komparator lesen
Callback-Funktionen	

`void` **`HAL_COMP_TriggerCallback`**`(COMP_HandleTypeDef* hcomp);`	Komparator-Trigger-Callback

DAC

Polling

`HAL_StatusTypeDef` **`HAL_DAC_Start`**`(DAC_HandleTypeDef* hdac, uint32_t Channel);`	DAC starten
`HAL_StatusTypeDef` **`HAL_DAC_SetValue`**`(DAC_HandleTypeDef* hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data);`	DAC-Ausgabe-Wert setzen

DMA

`HAL_StatusTypeDef` **`HAL_DAC_Start_DMA`**`(DAC_HandleTypeDef*hdac, uint32_t Channel, uint32_t*pData, uint32_t Length, uint32_t Alignment);`	DAC starten, mit DMA

Definitionen

`Channel``: DAC_CHANNEL_1, DAC_CHANNEL_2`

`Alignment``: DAC_ALIGN_8B_R, DAC_ALIGN_12B_L, DAC_ALIGN_12B_R`

Callback-Funktionen

`void` **`HAL_DAC_ConvCpltCallbackCh1`**`(DAC_HandleTypeDef* hdac);`	Conversion Complete Callback (Ch1)
`void` **`HAL_DAC_ConvHalfCpltCallbackCh1`**`(DAC_HandleTypeDef* hdac);`	Half Complete Callback (Ch1)
`void` **`HAL_DAC_ErrorCallbackCh1`**`(DAC_HandleTypeDef* hdac);`	Fehler aufgetreten (Ch1)
`void` **`HAL_DAC_DMAUnderrunCallbackCh1`**`(DAC_HandleTypeDef* hdac);`	Daten nicht rechtzeitig geliefert (Ch1)
`void` **`HAL_DACEx_ConvCpltCallbackCh2`**`(DAC_HandleTypeDef* hdac);`	Conversion Complete Callback (Ch2)
`void` **`HAL_DACEx_ConvHalfCpltCallbackCh2`**`(DAC_HandleTypeDef* hdac);`	Half Complete Callback (Ch2)
`void` **`HAL_DACEx_ErrorCallbackCh2`**`(DAC_HandleTypeDef* hdac);`	Fehler aufgetreten (Ch2)
`void` **`HAL_DACEx_DMAUnderrunCallbackCh2`**`(DAC_HandleTypeDef* hdac);`	Daten nicht rechtzeitig geliefert (Ch2)

Hinweis: *Diese Übersicht ist als Hilfe bei der Programmierung eines STM32-Mikrocontrollers gedacht. Ihre Verwendung setzt grundlegendes Verständnis der verwendeten Mikrocontroller-Module und der Prinzipien der HAL-Bibliothek voraus.*