UM3329

User manual

How to use STSW-AKIDRV and an STM32 to control the ADC120/ADC1283 through SPI

Introduction

This user manual provides information on using the STSW-AKIDRV software.

The ADC120 and ADC1283 are both 8 channel 12-bit ADCs. The demo boards for both devices are STEVAL-AKI001V1 and STEVAL-AKI002V1, respectively.

The STM32 family of 32-bit microcontrollers developed by STMicroelectronics are based on the Arm Cortex-M processor architecture and are widely used in a variety of applications.

Serial peripheral interface (SPI) is a synchronous serial communication protocol.

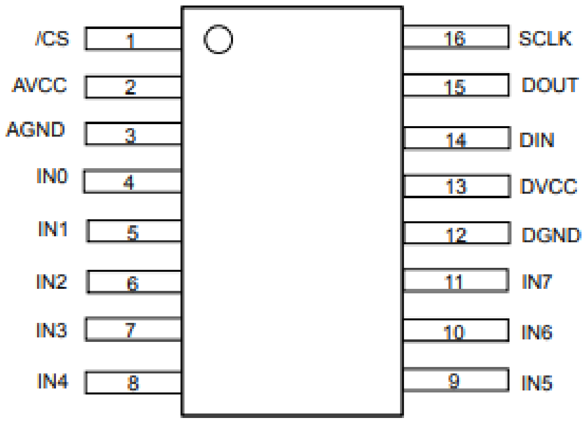
1 SPI configuration of your microcontroller

1.1 Pinout

As the ADC120 and ADC1283 are full duplex peripherals, the STM32 MCU must be used in full duplex controller mode to communicate with them.

Figure 1. ADC120/ADC1283 pinout

Figure 1. Pin description (top view)



The following table matches the STEVAL-AKI001V1 SPI pinout with the Nucleo STM32L476RG pinout.

Table 1. SPI pinout

Pin for SPI	STM32L476RG pinout to connect the STEVAL-AKI001V1
SCLK	PB13
MOSI	PB15
MISO	PB14
CS\	PB12

1.2 SPI speed

The frequency used by the microcontroller must match the different possible frequencies of the ADC120 and ADC1283.

Table 2. SPI communication speeds according to the device

	Minimum frequency	Maximum frequency
ADC120	800 kHz (50ksps)	16 MHz (1Msps)
ADC1283	800 kHz (50ksks)	3.2 MHz (200ksps)

1.3 Adress and ship select management

Both ADCs have 8 multiplexed inputs. These inputs can be selected thanks to several addresses.

Table 3. **Match between address and channel to read**

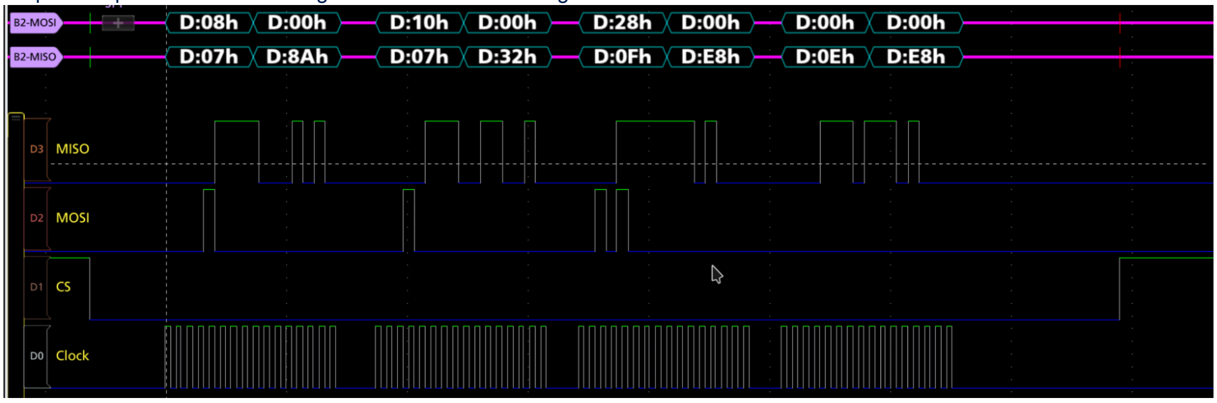
Input	Hexadecimal address
IN0	0x00
IN1	0x08
IN2	0x10
IN3	0x18
IN4	0x20
IN5	0x28
IN6	0x30
IN7	0x38

Regardless of the first input requested to read, the ADC120 and ADC1283 start by reading the channel 0.

The following figure provides an example showing several consecutive readings. During and between all readings, chip select must remain low.

Figure 2. **Example of reading several channels**

Chip select pin remains low during and between the readings



The following table shows that the first MISO payload is the channel IN0 digital conversion.

Table 4. **Explanation of reading steps, there is a shift between the ask of reading and the reading.**

Data on the bus	MOSI	08 00	10 00	28 00	00 00
	MISO	07 8A	07 32	0F E8	0E E8
Explanations	MOSI	Ask for IN1	Ask for IN2	ASK for IN5	
	MISO	Result of IN0	Result of IN1	Result of IN2	Result of IN5

Note: The chip select pin remains low during all the readings. If the chip select pin returns to high between two readings, the next value read will be the channel IN0 digital conversion.

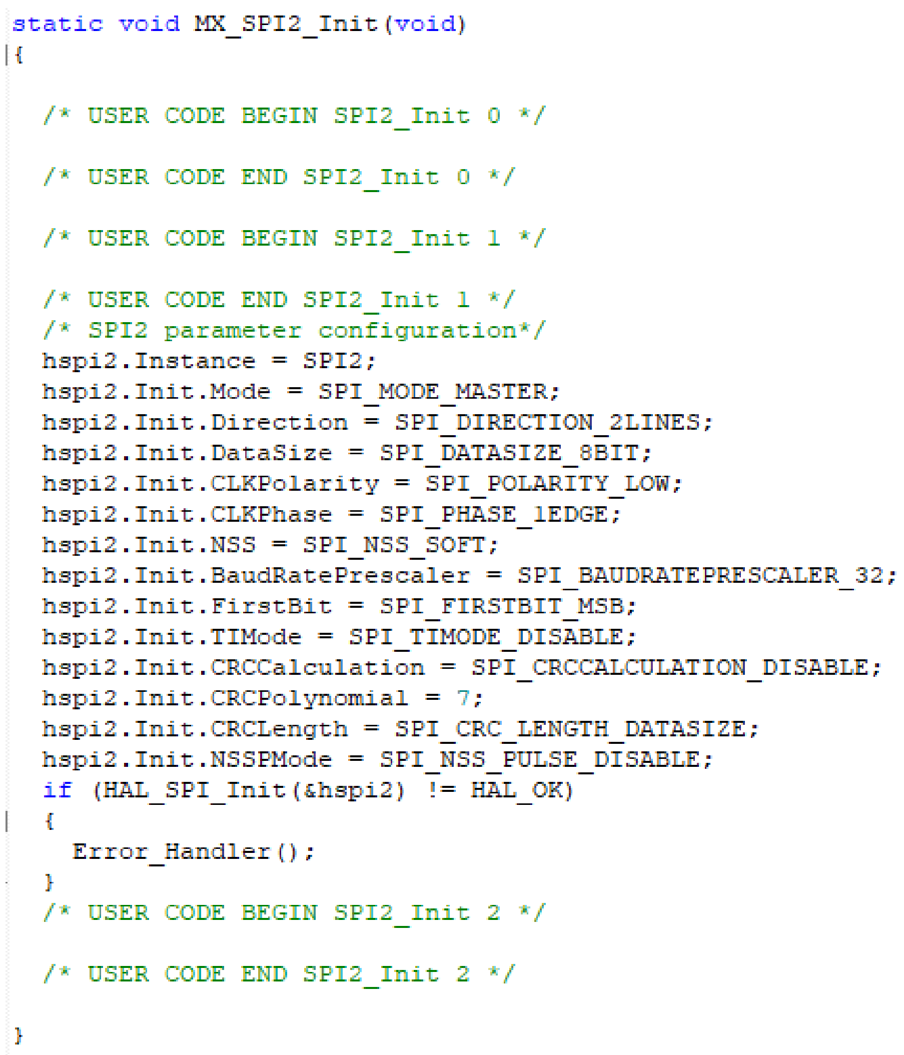
2 Use of the software

The STSW-AKIDRV software gives an example to use with the NUCLEO-L476RG.

2.1 Configuration of the SPI

The SPI configuration is performed through the `MX_SPI2_Init function`.

Figure 3. SPI initialisation function



```
static void MX_SPI2_Init(void)
{

  /* USER CODE BEGIN SPI2_Init 0 */

  /* USER CODE END SPI2_Init 0 */

  /* USER CODE BEGIN SPI2_Init 1 */

  /* USER CODE END SPI2_Init 1 */
  /* SPI2 parameter configuration*/
  hspi2.Instance = SPI2;
  hspi2.Init.Mode = SPI_MODE_MASTER;
  hspi2.Init.Direction = SPI_DIRECTION_2LINES;
  hspi2.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi2.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi2.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi2.Init.NSS = SPI_NSS_SOFT;
  hspi2.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_32;
  hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi2.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi2.Init.CRCPolynomial = 7;
  hspi2.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
  hspi2.Init.NSSPMode = SPI_NSS_PULSE_DISABLE;
  if (HAL_SPI_Init(&hspi2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN SPI2_Init 2 */

  /* USER CODE END SPI2_Init 2 */

}
```

The following table summarizes the values set to communicate with the ADC120/ADC1283.

Table 5. SPI initialisation values

Variable	Value	Explanation
hspi2.Instance	SPI2	Defines the instance of the SPI interface to use (SPI2).
hspi2.Init.Mode	SPI_MODE_MASTER	Defines the operating mode of the SPI interface as master mode.
hspi2.Init.Direction	SPI_DIRECTION_2LINES	Defines the data transfer direction using two lines (MOSI and MISO).
hspi2.Init.DataSize	SPI_DATASIZE_8BIT	Defines the size of the data to be transferred in 8 bits.

Variable	Value	Explanation
hspi2.Init.CLKPolarity	SPI_POLARITY_LOW	Defines the polarity level of the SPI clock as low.
hspi2.Init.CLKPhase	SPI_PHASE_1EDGE	Defines the phase of the SPI clock at the first transition.
hspi2.Init.NSS	SPI_NSS_SOFT	NSS (negative chip select) will be handled externally by a GPIO.
hspi2.Init.BaudRatePrescaler	SPI_BAUDRATEPRESCALER_32	Defines the prescaler of the SPI clock frequency as 32.
hspi2.Init.FirstBit	SPI_FIRSTBIT_MSB	Defines the data bit to be transferred first (MSB).
hspi2.Init.TIMode	SPI_TIMODE_DISABLE	Disables the TI (Transmit-Only) mode of the SPI interface.
hspi2.Init.CRCCalculation	SPI_CRCCALCULATION_DISABLE	Disables the CRC (Cyclic Redundancy Check) calculation for the transferred data.
hspi2.Init.CRCPolynomial	7	Defines the polynomial used for the CRC calculation.
hspi2.Init.CRCLength	SPI_CRC_LENGTH_DATASIZE	Defines the length of the CRC to the size of the transferred data.
hspi2.Init.NSSPMode	SPI_NSS_PULSE_DISABLE	Disables the pulse mode of the NSS line.

2.1.1 SPI speed according to the device

STSW-AKIDRV comes with a preset code for SPI communication.

In this example, the SPI speed can be modified in the `MX_SPI2_Init(void)` function.

In this function, one of the SPI init attributes is `hspi2.Init.BaudRatePrescaler`.

In the example, `hspi2.Init.BaudRatePrescaler` = SPI_BAUDRATEPRESCALER_32.

The following table provides the associated frequencies.

Table 6. Prescaler and the associated frequencies for a clock system (SYSCLK) of 80 MHz

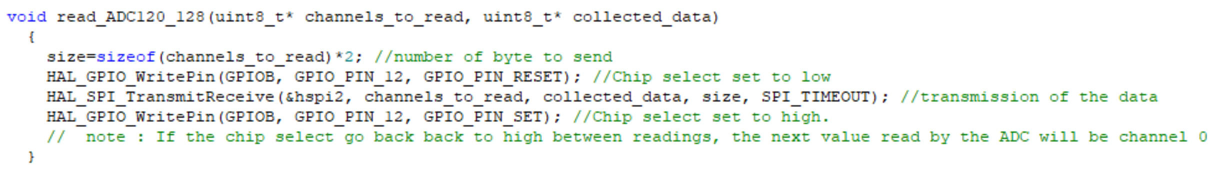
Prescaler	Frequency (MHz)
2	40 MHz
4	20 MHz
8	10 MHz
16	5 MHz
32	2.5 MHz
64	1.25 MHz
128	0.625 MHz
256	0.3125 MHz

2.2 Function to read the channels

Communication with ADC120 and ADC1283 is relatively straightforward. The simple function `read_ADC120_128()` allows reading one or several channels.

The STSW-AKIDRV comes with stm32 HAL (hardware abstraction layer) libraries to control the SPI communication and GPIOs.

Figure 4. read_ADC120_128() function example



```
void read_ADC120_128(uint8_t* channels_to_read, uint8_t* collected_data)
  {
    size=sizeof(channels_to_read)*2; //number of byte to send
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_RESET); //Chip select set to low
    HAL_SPI_TransmitReceive(&hspi2, channels_to_read, collected_data, size, SPI_TIMEOUT); //transmission of the data
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_SET); //Chip select set to high.
    //  note : If the chip select go back back to high between readings, the next value read by the ADC will be channel 0
  }
```

This function takes in two parameters:

- `channels_to_read`, which is a pointer to an array of uint8_t data type that specifies the channels to read from the ADC.
- `collected_data`, which is a pointer to an array of uint8_t data type that stores the collected data.

The function first calculates the number of bytes to send based on the size of the `channels_to_read` array and sets the chip select pin to low to activate the ADC.

It then uses the SPI protocol to transmit and receive data to and from the ADC.

Once the data are collected, the chip select pin is released to high again using `HAL_GPIO_WritePin` function.

Revision history

Table 7. Document revision history

Date	Version	Changes
20-Jun-2024	1	Initial release.

Contents

List of figures

List of tables