Here is a C function to perform a SPI transaction to read the temperature from a DS1722 sensor using CMSIS register-level access for an STM32L432KC.

This code assumes:

1. You have already initialized the SPI peripheral (e.g., `SPI1`) by enabling its clock in `RCC`, setting `SPE` (SPI Enable) in `SPI_CR1`, and configuring the `MSTR` (Master) bit, baud rate, `CPOL`, and `CPHA` settings.
2. The SPI data size is set to 8-bit (in `SPI_CR2`).
3. You have configured the associated GPIO pins (SCK, MISO, MOSI) for their SPI alternate functions.
4. You have configured the Chip Select (CS) pin as a standard GPIO output.

DS1722 Temperature Read Function (CMSIS)

```
#include "stm32l432xx.h"

// --- Define your SPI and CS pin configuration ---
// This example uses SPI1 and PA4 for CS
#define DS1722_SPI_INSTANCE   SPI1
#define DS1722_CS_GPIO_PORT   GPIOA
#define DS1722_CS_GPIO_PIN    GPIO_PIN_4

/**
 * @brief  Transmits and receives a single byte via SPI.
 * @note   This is a blocking function.
 * @param  SPIx: The SPI peripheral (e.g., SPI1).
 * @param  txData: The byte to send.
 * @return The byte received from the SPI slave.
 */
static inline uint8_t spi_transmit_receive_byte(SPI_TypeDef* S
    // Wait until the Transmit buffer is empty (TXE)
    while (!(SPIx->SR & SPI_SR_TXE));

    // Write the data to the Data Register to start the transa
    // Note: We cast to uint8_t* to force an 8-bit write.
    *(volatile uint8_t *)&SPIx->DR = txData;
    // Wait until the Receive buffer is not empty (RXNE)
    while (!(SPIx->SR & SPI_SR_RXNE));

    // Read the Data Register to get the received data and cle
    // Note: We cast to uint8_t* to force an 8-bit read.
    return *(volatile uint8_t *)&SPIx->DR;
}

/**
 * @brief  Reads the raw 16-bit temperature from the DS1722 se
 * @note   This function performs a 3-byte SPI transaction as
 * @return A 16-bit signed integer representing the raw temper
 * The LSB is 0.0625°C. The value is sign-extended.
 */
int16_t DS1722_ReadTemperature(void) {
    uint8_t temp_msb;
    uint8_t temp_lsb;
    int16_t raw_temperature;

    // 1. Pull Chip Select (CS) LOW to begin the transaction
    // We use the BRR (Bit Reset Register) for an atomic low o
    DS1722_CS_GPIO_PORT->BRR = DS1722_CS_GPIO_PIN;

    // 2. Send the command to read the Temperature MSB registe
    // The byte received during this first transfer is ignored
    spi_transmit_receive_byte(DS1722_SPI_INSTANCE, 0x01);

    // 3. Send the command to read the Temperature LSB registe
    // The byte received *during this transfer* is the Temp MS
    temp_msb = spi_transmit_receive_byte(DS1722_SPI_INSTANCE,

    // 4. Send a dummy byte (e.g., 0x00) to clock out the LSB
    // The byte received *during this transfer* is the Temp LS
    temp_lsb = spi_transmit_receive_byte(DS1722_SPI_INSTANCE,

    // 5. Wait for the SPI peripheral to be no longer busy
    while (DS1722_SPI_INSTANCE->SR & SPI_SR_BSY);

    // 6. Pull Chip Select (CS) HIGH to end the transaction
    // We use the BSRR (Bit Set/Reset Register) for an atomic
    DS1722_CS_GPIO_PORT->BSRR = DS1722_CS_GPIO_PIN;
    // 7. Combine the MSB and LSB into a 16-bit signed value
    raw_temperature = (int16_t)((temp_msb << 8) | temp_lsb);

    // Note: The DS1722 provides a 12-bit signed value, which
    // sign-extended to 16 bits in this 'raw_temperature' var
    // To convert to floating-point degrees Celsius, use:
    // float temperature_C = (float)raw_temperature * 0.0625f

    return raw_temperature;
}
```