What are header files?

They are separate files with the .h extension that contain declarations of functions, variables, and other elements that can be shared across multiple source files in a program.

They act as "interfaces" that provide information about code components without exposing their full implementation.

They promote code organization, reusability, and maintainability

File.h (prototype of used functions)

```
void begin(void) ;

void setDataMode (int, int ) ;

void setClockDivider(unsigned int );

void setBitOrder(int ) ;
int transfer(int );

void end(void) ;

void beginTransaction(int ,int,unsigned int, int);

void endTransaction(void) ;
```

File.c (body of the used functions)

void begin ()

```
{ enable_SPI _clock();
  SPI_GPIO_config();
  slaveManagementMode();
  SPI_enable();
}
```

void setDataMode(int mode, int clockMode){

```
  if(mode == 8)
 clear_bit(SPI1_CR1 ,11) // clear bit 11
     else if (mode == 16)
set_bit(SPI1_CR1 ,11);  // set bit 11

     switch (clockMode)
     {
       case 0:
        clear_bit(SPI1_CR1 , 1); CPOL
        clear_bit(SPI1_CR1 , 0); // (CPHA)
       break;

       case 1:
        clear_bit(SPI1_CR1 , 1);  // CPOL
        set_bit(SPI1_CR1 , 0); // (CPHA)
       break;

       case :
        set_bit(SPI1_CR1 , 1); // CPOL
        clear_bit(SPI1_CR1 , 0); // (CPHA)
       break;

       case 3:
        set_bit(SPI1_CR1 ,1); // (CPOL)
```

```
      set_bit(SPI1_CR1 ,0); // (CPHA)
      break;
    }
  }
```

void setClockDivider(unsigned int divider){

```
switch (divider)
      {
       case 2:              // setprescaler to 2
        clear_bit(SPI1_CR1 ,3);
        clear_bit(SPI1_CR1 ,4);
        clear_bit(SPI1_CR1 ,5);
       break;

       case 4:       // setprescaler to 4
        set_bit(SPI1_CR1 ,3);
        clear_bit(SPI1_CR1 ,4);
        clear_bit(SPI1_CR1 ,5);
       break;

       case 8:                    // setprescaler to 8
        clear_bit(SPI1_CR1 ,3);
        set_bit(SPI1_CR1 ,4);
        clear_bit(SPI1_CR1 ,5);
       break;
```

```
 case 16:                    // setprescaler to 16
  set_bit(SPI1_CR1 ,3);
  set_bit(SPI1_CR1 ,4);
  clear_bit(SPI1_CR1 ,5);
 break;

 case 32:                    // setprescaler to 32
  clear_bit(SPI1_CR1 ,3);
  clear_bit(SPI1_CR1 ,4);
  set_bit(SPI1_CR1 ,5);
 break;

 case 64:                    // setprescaler to 64
  set_bit(SPI1_CR1 ,3) ;
clear_bit(SPI1_CR1 ,4);
  set_bit(SPI1_CR1 ,5);
 break;

 case 128:                    // setprescaler to 128
  clear_bit(SPI1_CR1 ,3);
  set_bit(SPI1_CR1 ,4);
  set_bit(SPI1_CR1 ,5);
 break;

 case 256:                    // setprescaler to 256
  set_bit(SPI1_CR1 ,3);
```

```
        set_bit(SPI1_CR1 ,4);
        set_bit(SPI1_CR1 ,5);
       break;

     }
  }
```

void setBitOrder(int byteOrder){

```
   switch (byteOrder)
   {
     case 0:
      set_bit(SPI1_CR1 ,7); //LSB transmitted first
     break;
     case 1:
      clear_bit(SPI1_CR1 ,7); //MSB transmitted first
      break;
   }
  }
```

int transfer(int data)

```
 { int receivedData ;
   set_bit(SPI1_CR1 ,10);

   while (~(SPI1_SR & (1 << 1))) {/* Wait until transmit buffer is empty (TXE flag set)*/}
   while ((SPI1_SR & (1 << 7))) {/* Wait fot busy bit to reset */}
```

```
    // Write data to be sent into the data register
    SPI1_DR = data;

    // Wait until RX buffer is full (wait for receive data)
    while (!(SPI1_SR & (1 << 0))) {/* Wait until receive buffer is full (RXNE flag set)*/}
    while ((SPI1_SR & (1 << 7))) {/* Wait fot busy bit to reset */}

    // Read received data from data register
     receivedData = SPI1_DR;

    // Return received data
    return receivedData ;
}
```

void end() {

```
    // Wait until RXNE = 1 to receive the last data
    while (!(SPI1_SR & (1 << 0))) {/* Wait until RXNE flag is set (last data received)*/}

    // Wait until TXE = 1
    // Wait until BSY = 0
    while (SPI1_SR & (1 << 7)) {/* Wait until BSY flag is cleared (SPI not busy)*/}
   // Disable SPI (SPE = 0)
     clear_bit(SPI1_CR1 ,6);

    // Disable peripheral clock
    clear_bit(RCC_APB2ENR ,12); // Disable SPI1 clock
```

}

void beginTransaction(int byteOrder,int dataMode,unsigned int baudRate, int clockMode){

```
 //enable SPI1 clock
     set_bit(RCC_APB2ENR ,12);
 // Set byte order (MSB or LSB first)
     setBitOrder(byteOrder);

 // set data mode (8_bit or 16_bit mode) and clock mode
     setDataMode(dataMode,clockMode)

   //setting baud rate
     setClockDivider(baudRate); // Set BR bits to divide the clock

}
```

void endTransaction(){

```
    // Wait until RXNE = 1 to receive the last data
   while (!(SPI1_SR & (1 << 0))) {/* Wait until RXNE flag is set (last data
received)*/}

   // Wait until TXE = 1
   while (!(SPI1_SR & (1 << 1))) {/* Wait until TXE flag is set*/}

   // Wait until BSY = 0
   while (SPI1_SR & (1 << 7)) {/* Wait until BSY flag is cleared (SPI not busy)*/}
```

```
  // Disable SPI (SPE = 0)
    clear_bit(SPI1_CR1 ,6);

}
```