Problem Statement: Emulating data transfer in C

Task Requirements: You are supposed to emulate a data transfer using one of the following communication protocols: SPI, I2C, UART, CAN. Since the program will run on your machine locally, the data transfer is printed out on the command line instead of an actual data transmission.

The example shown below uses SPI protocol in which for a given input text (which could be of any length), the program converts it into binary and prints out three SPI lines: *Chip Select* (CS), *Clock* (CLK) and *Master Out Slave In* (MOSI) for each 20-byte block (as the data is sent in 20-byte blocks). MOSI in this case represents the input text provided to the program. For the fourth SPI line, *Master In Slave Out* (MISO), the program converts the binary representation back into ASCII and prints it out. During the process, the program should also take into account whether to use Most Significant Bit (MSB) or Least Significant Bit (LSB) first representation when converting into binary and vice versa.

In addition to the sender and the receiver, the monitor will be responsible to start the transmission process and will receive events, in the form of callbacks, from the sender/receiver about the transmission progress. When the transmission from the sender finishes, the monitor should then send a signal to the receiver to start its transmission (this transmission contains the data received from the sender).

In total, the program should have three separate files containing the functionality of the sender, the receiver, and the monitor (this can be the main file) respectively.

CHALLENGE: For the implementation, you are not allowed to use:

1. Global variables,
2. Variable Length Arrays (VLAs), and
3. Any library (header) other than the standard C libraries (such as stdio)

The example below uses SPI protocol for communication between the sender and the receiver to show you how the output could look like. However, in your implementation there should be additional output from the monitor printing the status of the transmissions (the progress as well as events like the completion of data transfer).

```
Input data: Hello world

CS:   110 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 011
CLK:      01010101 01010101 01010101 01010101 01010101 01010101 01010101 01010101 01010101 01010101 01010101
MOSI:     00010010 10100110 00110110 00110110 11110110 00000100 11101110 11110110 01001110 00110110 00100110

Data read (MISO): Hello world

--lsb first--

Input data: Hello world

CS:   110 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 011
CLK:      01010101 01010101 01010101 01010101 01010101 01010101 01010101 01010101 01010101 01010101 01010101
MOSI:     01001000 01100101 01101100 01101100 01101111 00100000 01110111 01101111 01110010 01101100 01100100

Data read (MISO): Hello world
```

Image explanation:

Input data: Represents a chunk of the input text (here, it is “Hello world” as an example)

CS: Represents the chip select line. It is active low in this case and therefore is high all the time except during a data transfer. Hence the first and the last two bits are shown as high in the output and the rest as low during data transfer.

CLK: Represents the clock line. It toggles between high and low for each bit during the data transfer.

MOSI: Represents the binary representation of the input text ("Hello world" in case of this example) transferred from the sender to the receiver. Depending on the selected representation, MOSI output is MSB-first (the top part of the image), otherwise each byte is reversed to represent LSB-first representation (the bottom part of the image).

MISO: Represents the assembled data on the receiver side, i.e., the binary representation converted back to ASCII. This conversion takes into account the selected representation (MSB- or LSB-first).