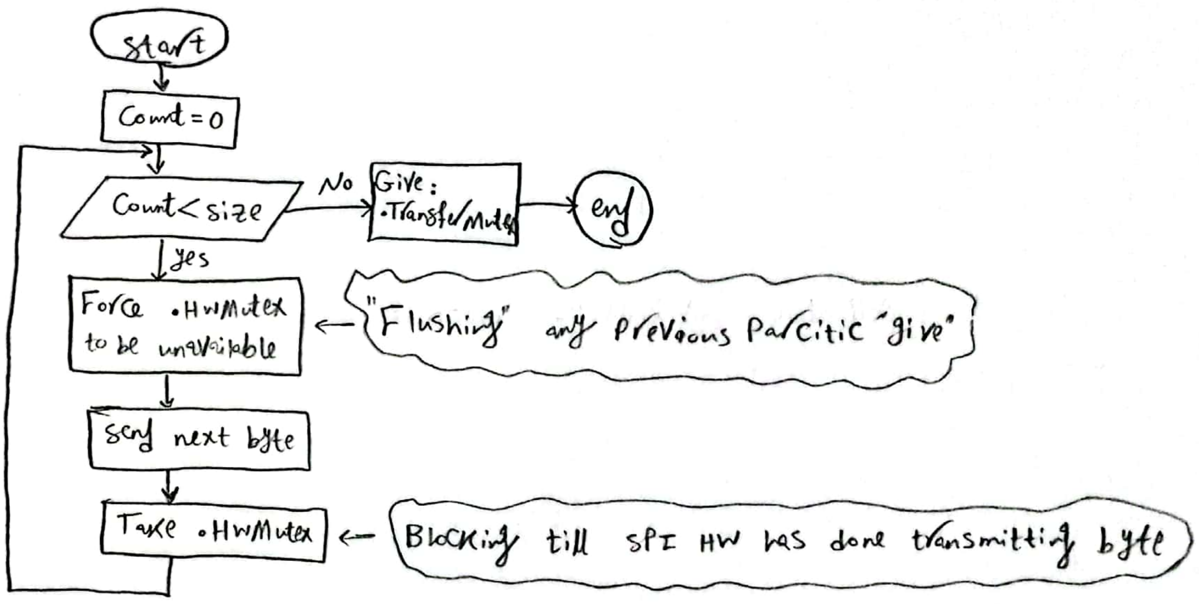
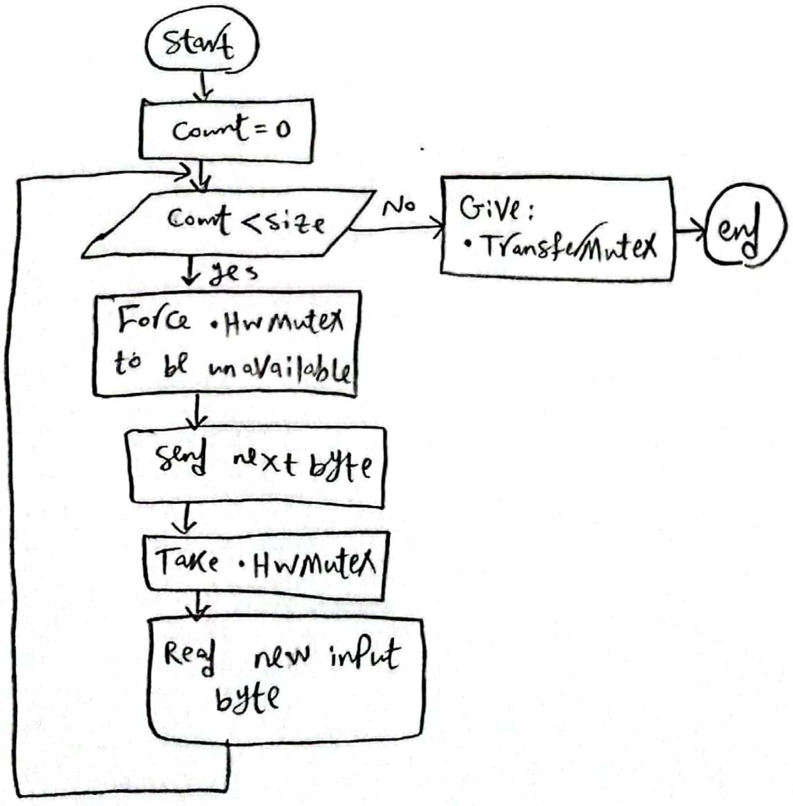
SPI_V1.0

send :

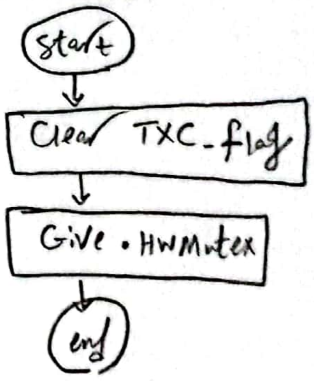


start
Count = 0
Count < size
No
Give : .TransferMutex
end
yes
Force .HwMutex to be unavailable
"Flushing" any previous Parcitic "give"
send next byte
Take .HwMutex
Blocking till SPI HW has done transmitting byte

Transceive :



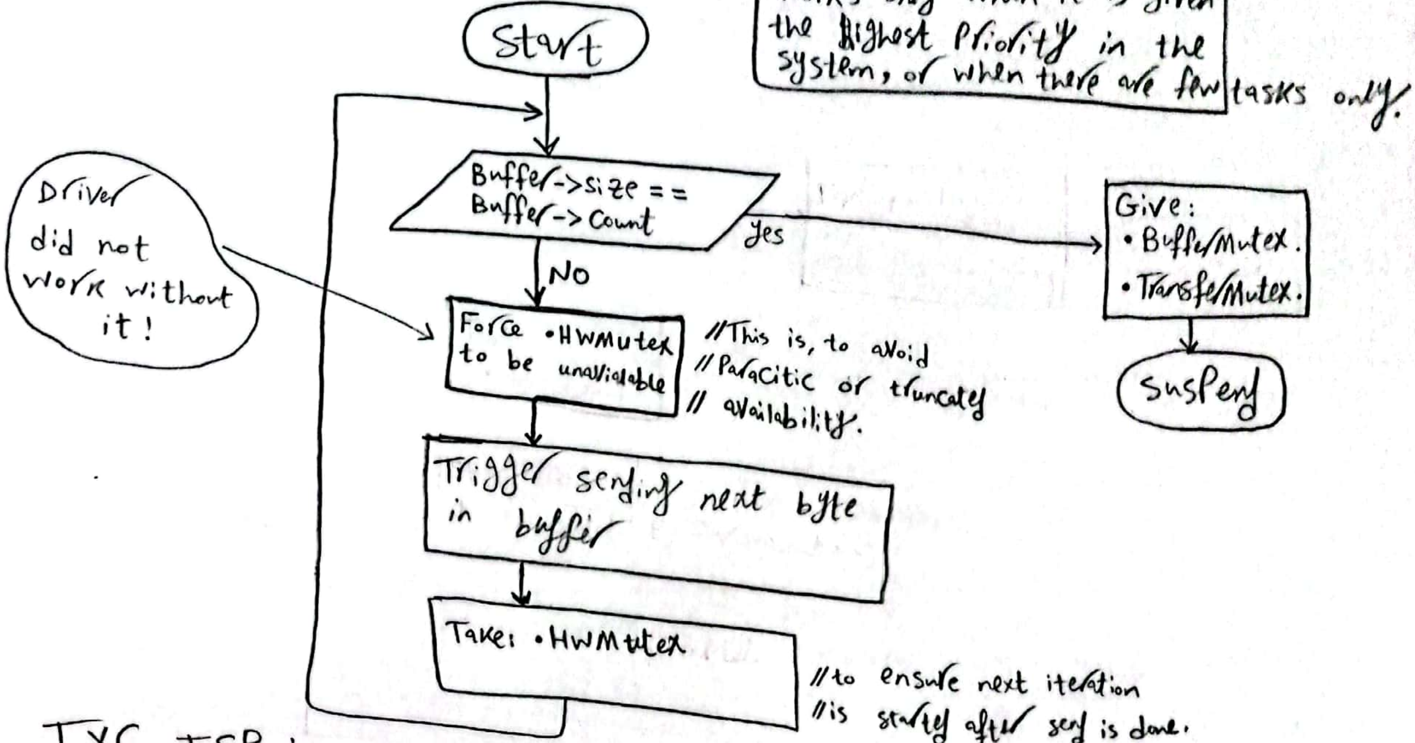
TXC_ISR :



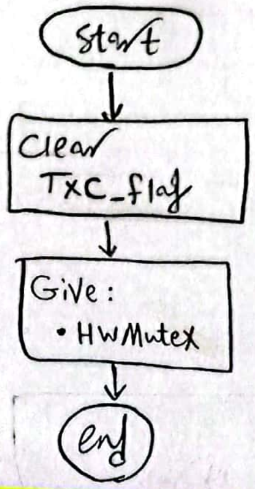
SPI V1.1 (Transmitting only)

Unit_task:

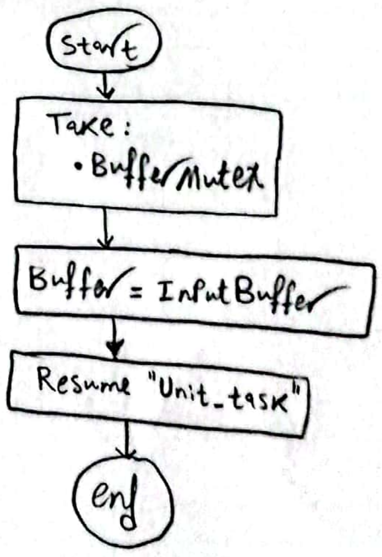
Delayed!
Works only when it is given the highest priority in the system, or when there are few tasks only.


Start
Buffer->size == Buffer->Count
Yes
Give: • BufferMutex. • TransferMutex.
Suspend
No
Driver did not work without it!
Force •HWMutex to be unavailable
//This is, to avoid //Paracitic or truncated // availability.
Trigger sending next byte in buffer
Take: •HWMutex
//to ensure next iteration //is started after send is done.

TXC_ISR:



Send:



V1.0 vs. V1.1

* one question that may come in mind: "why not sending and blocking in the task that uses SPI?".

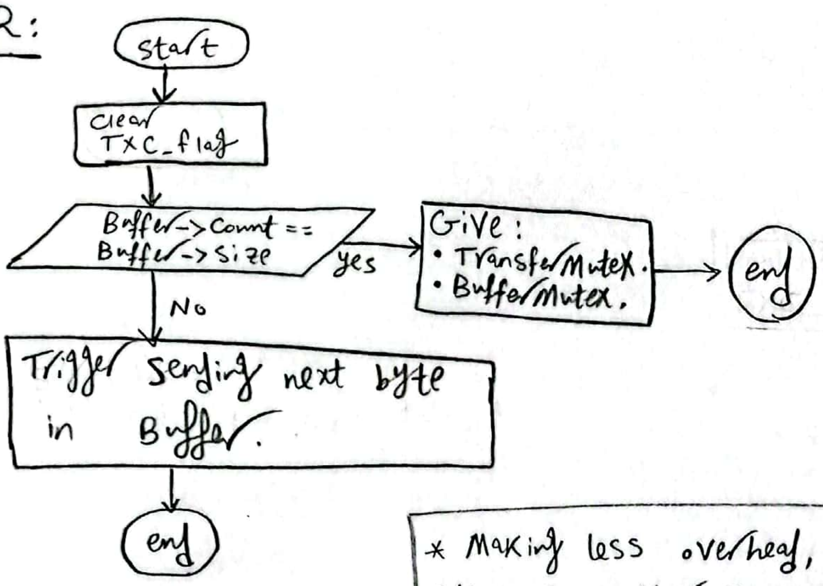
• This is done to prevent un-nescissary task blocking.

* For example: if it triggers sending and then does something that's independent from the sending, there's no need to block.

SPI V1.2 (transmitting only)

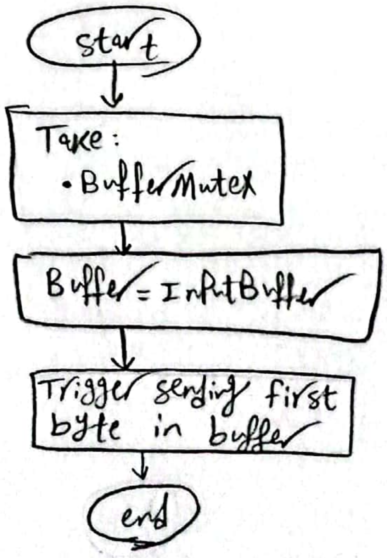
* As noticed in the SPI V1.1, "Unit_task" is not much of an overhead. Hence, it's a waste of time to use mutex Communication with ISR, Instead, let the ISR do the whole job.

TXC_ISR:



start
Clear TXC_flag
Buffer->Count == Buffer->Size
yes
Give:
• TransferMutex.
• BufferMutex.
end
No
Trigger sending next byte in Buffer.
end

send :



* Making less overhead, still doesn't mean V1.2 is better than V1.1. Because, V1.2 make SPI transmitting of a very high Pri--ority (Remember that lowest Priority ISR is higher in Priority than the highest Priori task).

* Hence, V1.2 is better only if the system design aims to give SPI transmission the highest execution Priority, or at least tolerates it.

■ Important note for SPI when using STM32:

* STM32 does not have Transfer_Complete_Interrupt_Flag !!

* "TXE" does not mean that transfer has been Completed! it means that data are just Copied from Tx-Register to the shift-register and is being Currently transferred.

* • As RXNE_flag is set after data has been Copied from the shift-register to RX-register,

• And Knowing that this would occur after transmitting the data from Tx → shift register → out-Pin fully,

• Hence, RXNE_flag Could be used as Tranfer_Complete_flag as well.