Inner Range Embedded Programming Test

Background:

An imaginary little endian 16 bit processor has a (very fast) UART. The processor hardware is capable of reading 8, 16 or 32 bit data on byte boundaries, and addresses memory in bytes

This UART has two memory mapped 16 bit registers, a control/status register and a data register.

The Control / Status register is at a byte address location 0x80000120

The Status register has contains 4 important bits:

Bit 0 – RX Register Not Empty (Read Only) – an rx data byte is ready to be read

Bit 1 – TX Register Not Full (Read Only) – There's space for another tx data byte.

Bit 2 – RX Error (Read Only)

Bit 13 – TX Enable (Read/Write).

Bit 14 – RX Enable (Read/Write).

Bit 15 – Interrupt Enable (Read/Write).

The Data register is at a byte address of 0x80000122.

The most recently received byte can be read from the Data register

Bytes to be transmitted are stored in the data register.

There is a single interrupt which (if enabled) interrupts on Status register:

IRQ = (Bit 0 && Bit 14) || (Bit 1 && Bit 13)

Interrupts save all registers to the stack (other than the stack pointer).

The macros INTERRUPT_DISABLE() and INTERUPT_ENABLE() and IS_INTERRUPT_ENABLED() exist and manipulate a global interrupt mask in the way you would expect.

Your task:

Write the following module(s) in C:

1) An Interrupt handler that receives and transmits bytes to / from a queue as appropriate.
    a. Overflowing the receive queue should somehow flag an error to a caller, lose data, but not crash
    b. The transmit queue may be empty on an interrupt
    c. Please maintain a 32 bit counter that counts the number of bytes received since power-up.

   d. You can choose any interrupt handler syntax with which you are familiar. If you're not sure, we suggest GCC.

2) An initialization function that:
   a. Configures two queues of 256 bytes each, a transmit queue to send bytes via the UART and a receive Queue that receives bytes via the UART.
   b. Performs any other initialization required by your solution
   c. Starts the interrupt handler
3) Interface functions that :
   a. Write n bytes of data from a byte pointer to the queue. Do not make any assumptions about how often this function is called.
   b. A blocking function to read n bytes of data from the queue to a byte pointer.
   c. A non-blocking function to read up to n bytes of data from the queue to a byte pointer.
   d. Do not make any assumptions about how often this function is called.
   e. Read how many bytes are in the receive queue
   f. Read how many bytes are in the transmit queue
   g. Read the total number of bytes received
4) A test function that exercises the interfaces of the system.

   All functions should be thread-safe in a pre-emptive multitasking environment, and be callable with interrupts turned on or off by the caller.

You may define the interface between these functions and the rest of the system

Interrupt latency should be minimised.

In the interests of fairness we will not answer exploratory questions or offer further clues, but please feel free to make any assumptions necessary to complete the task elegantly, but do document any assumptions and the reasoning behind them.

The code should compile (except for any bits that can't as it's an imaginary processor after all).

Feel free to describe the decisions you made.