Bern University
of Applied Sciences

Exercise - 22

UART Interrupt Transmit Behavior

Andreas Habegger | Adrian Steiner
BTS4230 | Version 1.0.0 of 17.03.2025

Using the MCU as a sender of periodic messages is a common application. The communication method is usually the UART. But how does the MCU behave when several tasks want to send a message at the same time? How can we avoid this? These questions will be addressed in this exercise.

Objectives

- HAL UART Transmit without DMA or interrupt
- Base timer with interrupt
- Interrupted functions
- Blocking IRQ

Outcomes

- Understanding of blocking IRQ
- Interpreting the HAL UART Send function
- Use of function return values

Description

This exercise is about sending messages from the superloop and from an IRQ generated by a basic timer (TIM6, TIM7). The behaviour of the received messages is examined using these two sources. The `HAL_UART_Transmit()` function without interrupt or DMA is used as the transmit function.

The function HAL_UART_Transmit()

```
/**
  * @brief  Sends an amount of data in blocking mode.
  * @note   When UART parity is not enabled (PCE = 0),
  *         and Word Length is configured to 9 bits (M1-M0 = 01),
  *         the sent data is handled as a set of u16. In this case,
  *         Size must indicate the number of u16 provided through pData.
  * @param  huart Pointer to a UART_HandleTypeDef structure that contains
  *               the configuration information for the specified UART module.
  * @param  pData Pointer to data buffer (u8 or u16 data elements).
  * @param  Size  Amount of data elements (u8 or u16) to be sent
  * @param  Timeout Timeout duration
  * @retval HAL status
  */
HAL_StatusTypeDef HAL_UART_Transmit(
   UART_HandleTypeDef *huart,
   const uint8_t *pData,
   uint16_t Size,
   uint32_t Timeout)
```

For the delay time, HAL provides a macro to set it to the maximum value called `HAL_MAX_DELAY`. This value is used for this exercise, but you can of course also test it with other values.

Tasks

We need to create an initial scenario as described above to begin the exercise.

Initial situation

1. Create a new STMCubeMX project with the default configuration.
2. Initialise a simple timer with an interrupt with a frequency of $10\,\mathrm{Hz}$.
3. Set the interrupt priority as high as possible.
4. Send a message of at least 8 characters from the interrupt callback function.
5. Test your implementation. Do you receive the message on your computer at the defined frequency?

Tip

On your serial monitor, enable the timestamp of the received message to be displayed. Set the correct end-of-line character (`'\n'` or `'\r'`) and add it to your message. Do not forget to configure the base timer with interrupt capabilities.

6. In addition to the interrupt, send another message in the superloop with the same conditions to detect which task has sent the message.
7. Answer the following questions about this exercise:

Question

a. Do you receive the message from both tasks?

b. Are the messages complete?

c. Are both transmit functions called?

d. Why do you think they behave the way they do?

e. How could it be solved?

Status codes

The function will return a `HAL_StatusTypeDef` value. Study the HAL function used for this exercise.

HAL_StatusTypeDef

```
typedef enum {
  HAL_OK       = 0x00U,
  HAL_ERROR    = 0x01U,
  HAL_BUSY     = 0x02U,
  HAL_TIMEOUT  = 0x03U
} HAL_StatusTypeDef;
```

Study the HAL_UART_Transmit() function

8. Study the transmission function used and answer the following questions.

Question

a. Which return value do you receive for which behaviour?

b. Can you be sure that the message is sent when the function returns if you use the `HAL_MAX_DELAY` macro?

The new insights are now integrated into the exercise. Implement a return value check with a blocking while loop.

Check the return value

9. In both tasks, add a blocking wait with a while loop until the message has been sent completely.
10. Answer the following questions about this exercise:

Question

a. Are you receiving the message from both tasks?

b. Are the messages complete?

c. Are both tasks called?

d. What is their return value?

e. What is the problem with the blocking while loop?

Influence of the HAL_Delay() function

The behaviour with a small blocking delay in the superloop is checked and compared with the previous one. Test it with the two situations above.

HAL_Delay()

10. Add a small delay of $1\,\mathrm{s}$ to the superloop.
11. Answer the following questions about this exercise:

Question

a. Do you get the message from both tasks?

b. Are the messages complete?

c. Why do you think they behave the way they do?

d. Is this a good idea?