Using the PL011 UART transmit interrupt to write data when the FIFO is disabled

Application Note

Document number: ARM DAI 0256A

ARM®

Using the transmit interrupt to write data to the UART when the FIFO is disabled Application Note

Release information

The Change history table lists the changes made to this document.

Table 1 Change history

Date	Issue	Confidentiality	Change
March 2011	A	Non-Confidential	First release

Proprietary notice

Confidentiality status

This document is Non-Confidential. This document has no restriction on distribution.

Feedback on this application note

If you have any comments on content then send an e-mail to errata@arm.com. Give:

- the document title
- the document number
- the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

ARM web address

http://www.arm.com

Table of Contents

Table of Figures

1 Introduction

This document describes one software solution to a particular issue with the PL011 UART r1p5 whereby the Transmit Interrupt (UARTTXINTR) only fires under certain conditions. The conditions are valid, as is the current behavior of the PL011 UART; the behavior, though, is not necessarily what users might expect.

The Technical Reference Manual (TRM) describers the behavior of the PL011 UART but fails to describe precisely the behavior under the conditions described in this document.

1.1 Purpose

The purpose of the document is to explain in detail the behavior of the PL011 UART under certain operating conditions

1.2 Additional reading

This section lists publications by ARM and by third parties.

See Infocenter, http://infocenter.arm.com, for access to ARM documentation.

1.2.1 ARM publications

The following documents contain information relevant to this document:

- *PrimeCell UART PL011 Revision: r1p5 Technical Reference Manual* (ARM DDI 0183G)
- *ARM PrimeCell UART (PL011) Design Manual* (PL011 DDES 0000)
- *ARM PrimeCell UART (PL011) Integration Manual* (PL011 INTM 0000)

1.2.2 Other publications

The following documents list relevant documents published by third parties:

- Infrared Data Association®, *IrDA® Physical Layer Specification v1.1*, obtainable at `http://www.irda.org`
- Agilent Technologies, *Agilent IrDA Data Link Design Guide* (5988-9321E. March 2003), obtainable at http://www.agilent.com.

2 Description

This app note concerns the case where the UART is being used with FIFOs DISABLED although it is also relevant when the FIFOs are enabled. When the FIFOs are disabled, the UARTTXINTR fires every time the holding register becomes empty due to a bye being transmitted.

The transmit interrupt changes state when one of the following events occurs:

• If the FIFOs are **enabled** and the transmit FIFO is equal to or lower than the programmed trigger level then the transmit interrupt is asserted HIGH. The transmit interrupt is cleared by writing data to the transmit FIFO until it becomes greater than the trigger level, or by clearing the interrupt.
• If the FIFOs are **disabled** (have a depth of one location) and there is no data present in the transmitter's single location, the transmit interrupt is asserted HIGH. It is cleared by performing a single write to the transmit FIFO, or by clearing the interrupt.

To update the transmit FIFO you must:
• Write data to the transmit FIFO, either prior to enabling the UART and the interrupts, or after enabling the UART and interrupts.

Note
The transmit interrupt is based on a transition through a level, rather than on the level itself. When the interrupt and the UART is enabled before any data is written to the transmit FIFO the interrupt is not set. The interrupt is only set after written data leaves the single location of the transmit FIFO and it becomes empty.

2.1 Simple example

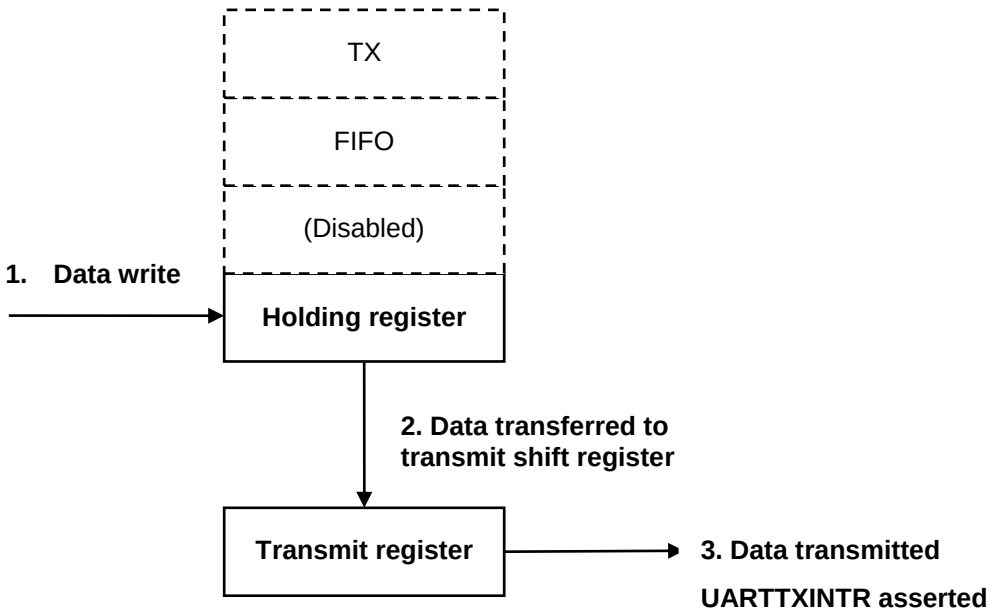
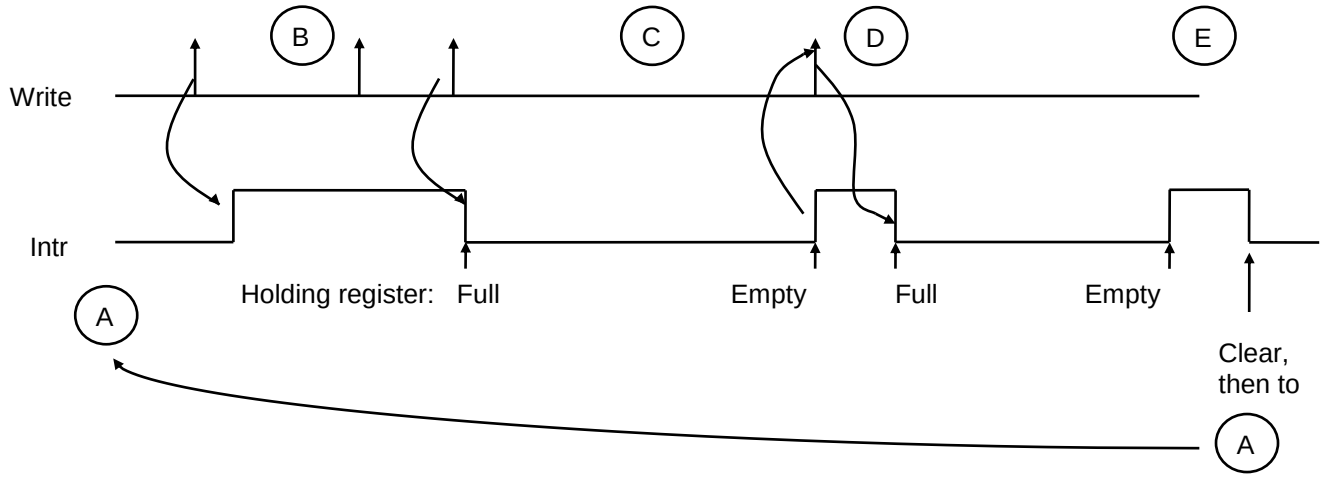


Figure 1 Normal data transmission – data write occurs whilst transmit shift register is in use

Under normal circumstances, when a byte of data is written to the UART for transmission, it first goes into the Transmit (TX) FIFO. If the FIFOs are disabled, then the FIFO is configured as a single-byte holding register. The transmit logic then moves this byte from the holding register and places it in the transmit shift register from where

it is transmitted. Under these circumstances the UARTTXINTR interrupt is asserted because the data in the holding register transitions from full to empty. The interrupt clear register is only written to at the end of the sequence when there is no data remaining to be transferred to the UART.

This simple interrupt-driven behaviour can be illustrated by the following sequence of events:



A. Quiescent state, write data to start. UARTTXINTR goes HIGH (special trigger condition)
B. UARTTXINTR HIGH: write data until UARTTXINTR LOW (holding register full)
C. Transmit data until UARTTXINTR goes HIGH (holding register empty, still transmitting the data that was just transferred)
D. Write data until UARTTXINTR goes LOW (holding register full)
E. Transmit data. UARTTXINTR goes HIGH, no data left, write Interrupt Clear Register. Back to A.

Figure 2 Simple interrupt-driven behavior

2.2 Corner-case examples

The precise behaviour of the UART depends on the timing between APB writes and the asynchronous character transmission, particularly if the interrupt is cleared using the interrupt clear register.

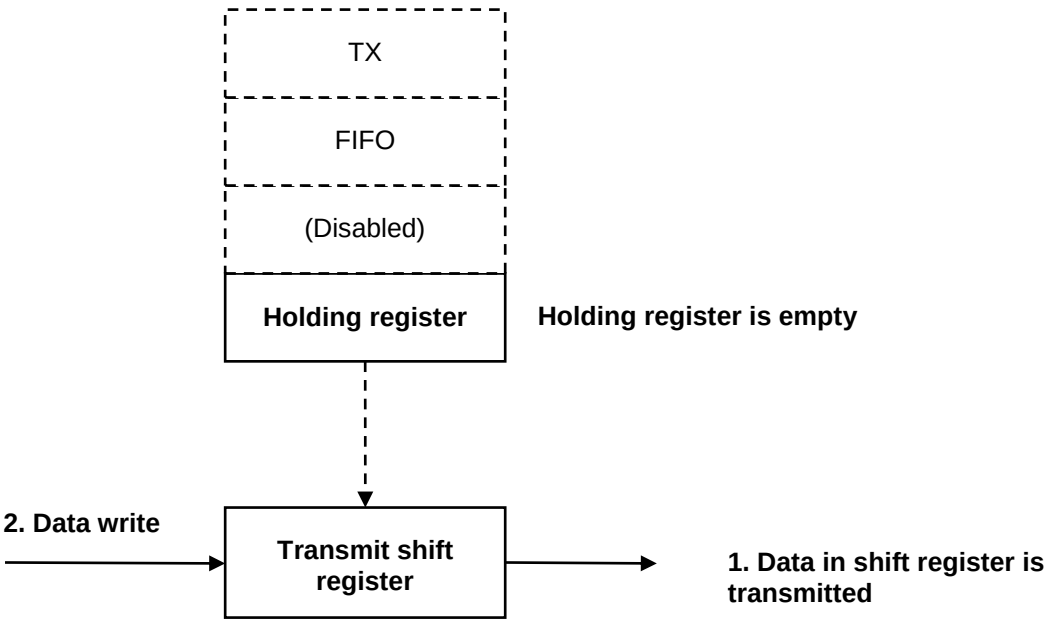


Figure 3 Data transfer from transmit shift register completed before data write

a. Figure 3 shows the following situation; the data in the transmit shift register is already transmitted before a new byte is written to the UART Data Register. The transmit shift register can therefore accommodate the new data and the holding register is not used. The written data is placed directly into the transmit shift register; hence, the UARTTXINTR ***remains unchanged***.

b. If the transmit shift register has not finished transmitting the character before the new character is written to the data register, the data is placed into the transmit holding register, and this process will *deassert* the UARTTX interrupt as shown in Figure 4.

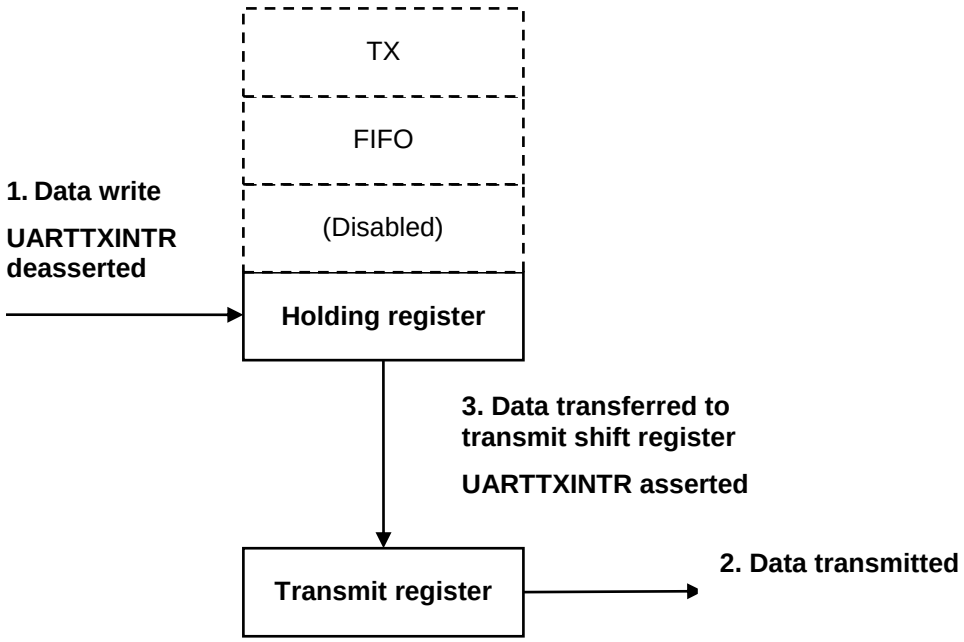


Figure 4 Data write to holding register before transmission of byte in transmit register is completed.

c. In one particular case, data is written to the data register at the same time as the transmit shift register becomes free. Under these circumstances, the holding register is bypassed. Since the holding register is not full the UARTTXINTR will not be cleared, and in the event that the interrupt had already been cleared, it will not be asserted when transmission completes. This is a result of the holding register not transitioning from full to empty..

As detailed in the TRM in section 2-23., the interrupt is generated by a transition from data in the holding register to no data in the holding register. In this specific case, the holding register is not being used.
The unusual condition which may cause confusion is the initial assertion of the interrupt when data is written to the data register when the transmit shift register is also empty. This is intended to start a sequence where assertion of the UARTTXINTR indicates that there is space in the holding register to accept more data.

The following table shows the behaviour of the UARTTX interrupt for a give set of starting conditions.

Notes:

1. The ***Starting Conditions*** are the state of the UARTTX interrupt and the contents of the Holding and Shift registers when the particular write sequence begins.
2. ***States*** are groups of events
3. ***Events*** are the numbered boxes and indicate the behavior of the UART for any given combination of starting conditions.
4. The greyed-out events are unattainable.
5. The event in **red*** should be noted as being of specific value at the start of a transmit sequence.

	Starting conditions				
State	**Interrupt**	**Registers**	**Write to UART Data Register**	**Transmission of character finishes**	**Write to UARTDR *and* transmission at *the same time***
A	**IRQ asserted**	**Data in holding register ONLY**	1	2	3
B		**Data in Shift register ONLY**	4 IRQ **Cleared**	5 TX register empties, IRQ remains **HIGH**	6 IRQ remains **HIGH**
C		**Data in both holding and Shift registers**	7	8	9
D		**Data in neither register**	10 IRQ Disabled, UART **Disabled**	11	12
E	**IRQ not asserted**	**Data in holding register ONLY**	13	14 Write before UART enabled, IRQ remains **LOW**	15
F		**Data in Shift register ONLY**	16 IRQ remains **LOW**	17 TX register empties, IRQ remains **LOW**	18 IRQ remains **LOW**
G		**Data in both holding and Shift registers**	19	20 IRQ **Asserted**	21 IRQ remains **LOW**
H		**Data in neither register**	22 **IRQ Asserted***	23	24

Table 2 shows the behavior of the UARTTX interrupt for a give set of starting conditions

2.3 Transmitting a single byte-at-a-time using BUSY

If it is required to transmit a sequence of discrete, single bytes, then there are no generic solutions using interrupts that can indicate when each byte has transmitted.

The BUSY flag (UARTFR[3]) can, however, be used to indicate when a byte has finished transmitting. It is asserted as soon as a byte is written to the FIFO, either in FIFO mode or as a holding register and will remain asserted until the byte has finished transmitting from the shift register.

If the FIFO/holding register is provided with single bytes, waiting between bytes until BUSY clears before writing another byte to the FIFO, BUSY can be reliably used as an indicator of transmission (figure 5). If the FIFO is continuously supplied with bytes, BUSY will remain set until the last byte has transmitted (figure 6). BUSY remains set while there is data in the FIFO/holding register or the shift register.

Since there is no external signal to indicate BUSY, a timer-driven interrupt service routine to poll the flag register should be used (see section 4 BUSY detection).

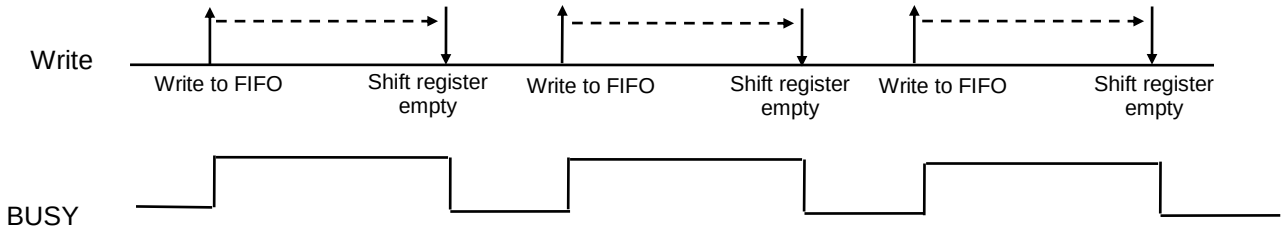


Figure 5 Single byte writes using BUSY flag

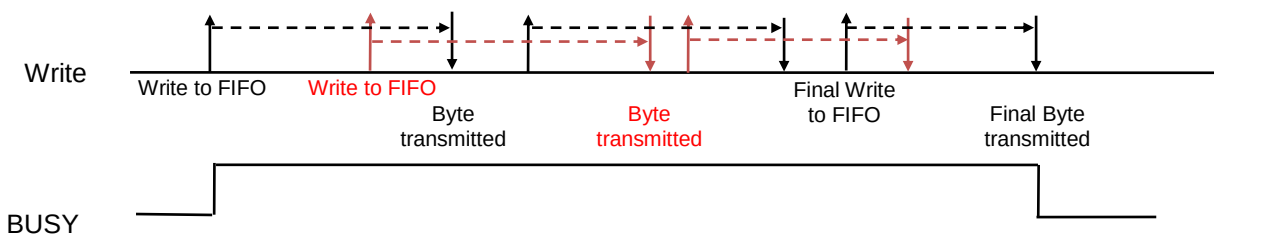


Figure 6 Multiple byte writes using BUSY flag

2.3.1 How the BUSY flag behaves

If the UART is idle, i.e. the holding register/FIFO and the shift register are both empty, BUSY is asserted when the holding register/FIFO is written. BUSY will then remain asserted until that byte has finished being transmitted and the shift register is empty again (figure 5).

If the shift register does not have time to empty before another byte is written to the holding register/FIFO, BUSY will remain asserted until such time as the UART becomes idle once more (figure 6).

2.3.2 BUSY and interrupts

BUSY is not directly correlated with UARTTXINTR. BUSY will assert when PL011 is idle and a byte is written to the holding register. UARTTXINTR is asserted according to the conditions shown in table 2 and typically indicates when a data byte transitions from the holding register to the shift register. The interrupt does **not** indicate when a byte has been transmitted.

2.3.3 Using BUSY to transmit data

BUSY can be used to determine when a byte has finished transmitting; figures 5 and 6 show how this is done. However, since BUSY is not associated with an interrupt, software will have to poll the BUSY flag in the status register. This can be done using a regular interrupt service routine associated with a timer (see section 4 BUSY detection).

The frequency of this timer needs to be such that transitions from 0 -> 1 and 1 -> 0 of the BUSY flag can be reliably detected. The UART can transmit anything between seven (1 start, 5 data, 0 parity and 1 stop bits) and twelve (1 start, 8 data, 1 parity and 2 stop) bits per byte. In the 'worst case' therefore, the polling frequency needs to be such that it can reliably detect the transition in the BUSY bit when only seven bits are used.

A sampling frequency of *at least twice the character frequency** should be enough to capture this transition, where:

Character frequency = baud rate/bits per character

So, assuming a baud rate of 921,600 and a character of 7 bits (worst case),

Character frequency = 921,600/7 = 131,658 characters per second

BUSY bit sampling frequency = 2 x character frequency = 2 x 131,658 = **263,316 samples per second**

= every **3.797** µs

*Theoretically, based on Nyquist's Theorem

3 Software Driver Solution

The following code structure can be used as an example of how to use the TXFF interrupt to control transfer of data to the UART and also allow that interrupt to be cleared during the interrupt handler:

Note:

It will need to be modified to fit into the software framework in use.

```
unsigned int n = 18 ;                 // Size of the data buffer
char data_buffer[n] ;                 // Data buffer
char *data_buffer_base = data_buffer; // Base address of data buffer

uart_update()
{
  // Loop until TXFF = 1 OR *data_buffer = 0
  while((((volatile *)uart_flag_register && TXFF) != 1) & (*data_buffer == 0))
  {
    if((*uart_flag_register && TXFF) == 0)     // Is TX FIFO full?
    {                                          // No
      if (*data_buffer  != 0)                  // Is there data in the buffer to be sent?
      {                                        // Yes
        *uart_data_register = *data_buffer;    // Copy one byte of this data to the FIFO...

        // Check if the data_pointer has gone too far. Wrap if it has
        if(++data_buffer > (((data_buffer_base + (sizeof(data_buffer) * n)))))
          data_buffer = data_buffer_base ;
      }
    }
  }
}
```

Where:

n is the maximum size of the data buffer, in bytes (adjust to suit application)

data_buffer is a byte-addressable circular buffer containing the data to be written to the UART, NULL terminated

data_buffer_base is the base address of this data buffer

uart_flag_register is the UART's Flag Register, UARTFR. This must be declared as **volatile**.

TXFF is the Transmit FIFO Full flag , bit[5], within the UARTFR

uart_data_register is the UART's Data Register, UARTDR

uart_update must be called whenever the data_buffer is updated and when UARTTX is asserted. For instance, say it is required to print the string "Hello World". Software would add (concatenate) "Hello World" to any data already in data_buffer, ensuring the last character in data_buffer is a NULL.

4 BUSY detection

The following code can be used as a timer-driven interrupt service routine to detect the transitions on the BUSY bit and to send the next character while data is available.

Note:

It assumes an initial 0 -> 1 BUSY transition.

```
unsigned int n = 18 ;                 // Size of the data buffer
char data_buffer[n] ;                 // Data buffer
char *data_buffer_base = data_buffer; // Base address of data buffer


check_BUSY()
{
  static unsigned current_BUSY = 0 ;

  current_BUSY = ((volatile *)uart_flag_register_&& BUSY) ; // Get BUSY status

  if(current_BUSY == 0)
  {
    // Check for more data. Keep feeding the FIFO as long as there is data
    // available and space in the FIFO/holding register
    if(((volatile*)uart_flag_register && TXFF) == 0) // Space in the FIFO/holding
    {                                                // register
      if(*data_buffer != 0) // Data in buffer
      {
        *uart_data_register = *data_buffer; // Copy one byte of data to the FIFO...

        // Check if the data_pointer has gone too far. Wrap if it has
        if(++data_buffer > (((data_buffer_base + (sizeof(data_buffer) * n)))))
          data_buffer = data_buffer_base ;
      }
    }
  }
}
```

Where:

n is the maximum size of the data buffer, in bytes (adjust to suit application)

data_buffer is a byte-addressable circular buffer containing the data to be written to the UART, NULL terminated

data_buffer_base is the base address of this data buffer

uart_flag_register is the UART's Flag Register, UARTFR. This must be declared as **volatile**.

TXFF is the Transmit FIFO Full flag , bit[5], within the UARTFR

BUSY is the UART busy status flag, bit[3], within the UARTFR

current_BUSY is the current value of the BUSY status flag

uart_data_register is the UART's Data Register, UARTDR

5 Writing to the data buffer

The figure below illustrates how writing to the data_buffer should work:

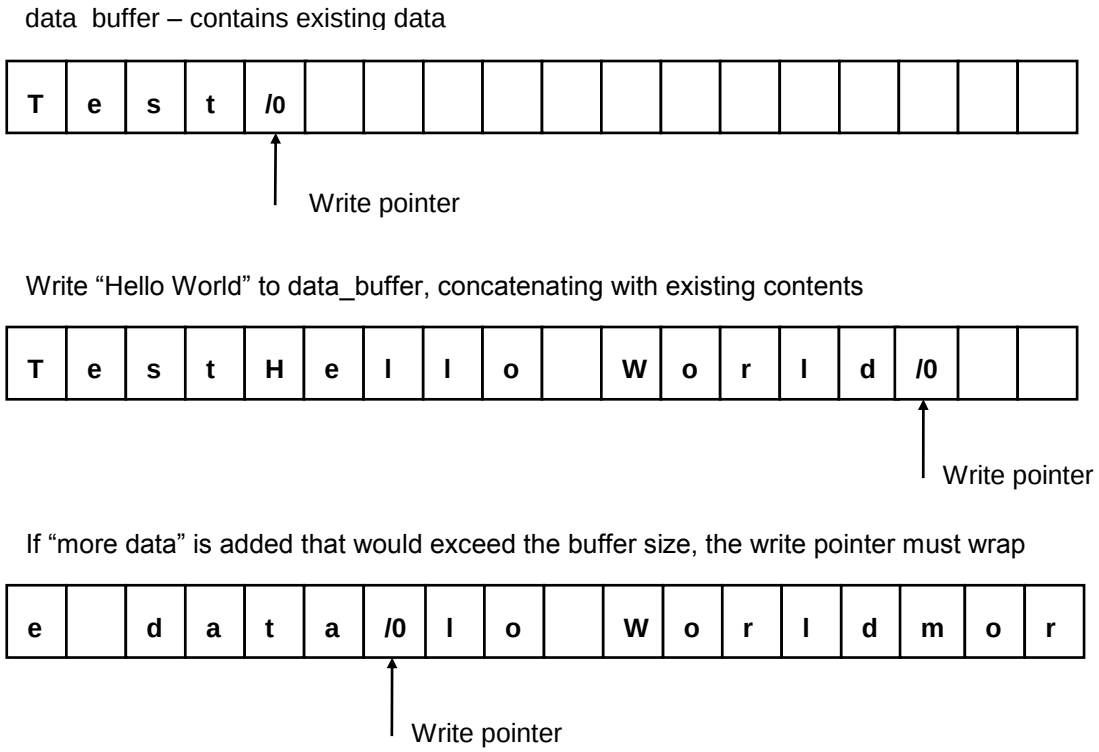


Figure 7 Writing to data_buffer

uart_update loops until EITHER the transmit holding register is empty (`*uart_flag_register && TXFF`) OR the data_buffer is empty (`*data_buffer == 0`).

Note:

It is important that the user considers the implications of a system interrupt occurring during the execution of uart_update, and so avoid any of the buffers overflowing.