



# **UART Transmitter and Receiver Macros**

**8-bit, no parity, 1 stop bit  
Integral 16-byte FIFO buffers**

**Virtex, Virtex-E, Virtex-II,  
Spartan-II and Spartan-IIE**

**Ken Chapman**

**Xilinx Ltd**

**October 2002**



# Limitations

**Limited Warranty and Disclaimer.** These designs are provided to you “as is”. Xilinx and its licensors make and you receive no warranties or conditions, express, implied, statutory or otherwise, and Xilinx specifically disclaims any implied warranties of merchantability, non-infringement, or fitness for a particular purpose. Xilinx does not warrant that the functions contained in these designs will meet your requirements, or that the operation of these designs will be uninterrupted or error free, or that defects in the Designs will be corrected. Furthermore, Xilinx does not warrant or make any representations regarding use or the results of the use of the designs in terms of correctness, accuracy, reliability, or otherwise.

**Limitation of Liability.** In no event will Xilinx or its licensors be liable for any loss of data, lost profits, cost or procurement of substitute goods or services, or for any special, incidental, consequential, or indirect damages arising from the use or operation of the designs or accompanying documentation, however caused and on any theory of liability. This limitation will apply even if Xilinx has been advised of the possibility of such damage. This limitation shall apply notwithstanding the failure of the essential purpose of any limited remedies herein.

This module is **not** supported by general Xilinx Technical support as an official Xilinx Product. Please refer any issues initially to the provider of the module.

Any problems or items felt of value in the continued improvement of these macros would be gratefully received by the author.

Ken Chapman  
Staff Engineer - Applications Specialist  
email: chapman@xilinx.com

The author would also be pleased to hear from anyone using KCPSM or KCPSM-II with these macros with information about your application and how these macros have been useful.



# Introduction

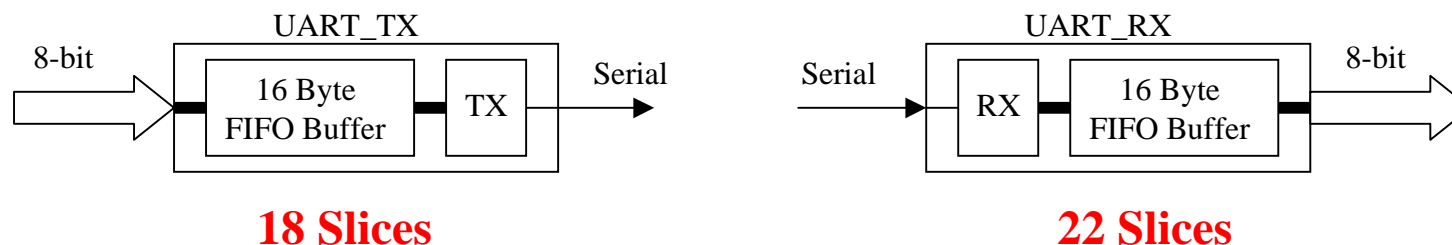
This package contains a pair of macros which have been highly optimised for the Virtex, VirtexE, Virtex-II, Spartan-II, and Spartan-IIE devices from Xilinx. The macros provide the functionality of a simple UART transmitter and simple UART receiver each with the fixed characteristics of:-

- 1 start bit
- 8 data bits (serially transmitted and received least significant bit first)
- No Parity
- 1 stop bit

As well as being able to use these macros as a pair to communicate with each other, they are also fully compatible with standard UART communication protocols such as to a PC (providing level shifting components are employed to generate RS232 signaling).

## What makes them Special?

Each macro also contains an embedded 16 byte FIFO buffer, and yet just look at the total size. Their small size makes them an ideal companion to the small KCPSM and KCPSM-II processor macros .....



Although standard baud rates from 9600 to can be supported, these macros are also capable of baud rates exceeding 10 M-bit/second offering an easy way to communicate data between Xilinx devices.

# Size and Performance

The following device resource information is taken from the ISE reports for the UART\_TX and UART\_RX macros in an XC2S50E device. The reports reveal the features that are utilised and the efficiency of the macro. The 18 and 22 'slices' reported by the map process in these case may reduce when greater packing is used to fit a complete design into a device.

## XST Reports

	uart_tx	uart_rx
LUT2	: 4	1
LUT3	: 3	2
LUT4	: 14	8
mult_and	: 3	
muxcy	: 6	3
xorcy	: 7	4
muxf5	: 2	
muxf6	: 1	
FD	: 1	3
FDE	: 5	21
FDR	: 1	1
FDRE	: 7	4
FDRS	: 1	
srl16e	: 9	27

## MAP Reports

### uart\_tx

Number of Slices: 18 out of 768 2%  
Total equivalent gate count for design: 1,452

### uart\_rx

Number of Slices: 22 out of 768 2%  
Total equivalent gate count for design: 3,775

## TRACE Report

Device,speed: xc2s50e,-7 (PRELIMINARY 1.14 2002-08-21)  
Clock to Setup on destination clock  
uart\_tx 6.309  
uart\_rx 6.342

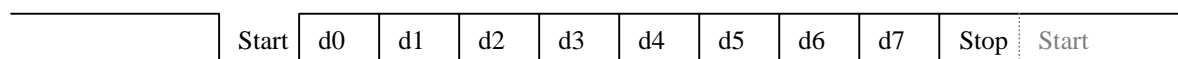
158 MHz



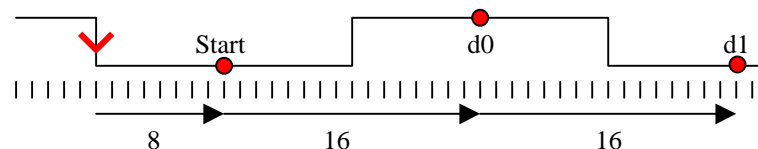
# UART Operation

An Asynchronous Receiver and Transmitter means by its very nature that the transmitter and receiver are not synchronised. However, they do both utilise a timing reference which is of a suitable tolerance to allow the serial transfer of each byte of data.

The data is transmitted serially LSB first at a given bit rate (BAUD rate) which is known by the transmitter and receiver. Since the transmitter can start sending this data at any time, the receiver needs a method of identifying when the first (LSB) is being sent. This is achieved by the transmitter sending an active low start signal for the duration of one bit.



The receiver uses the falling edge of the start bit to begin an internal timing circuit. This timing is then used to sample the value of the serial input at a point which is approximately at the mid-position of each data bit. This is where the data should be most stable. After the last data bit (MSB) has been sampled, the receiver checks to see if the transmitted stop bit (high) is the value expected which helps confirm correct operation.



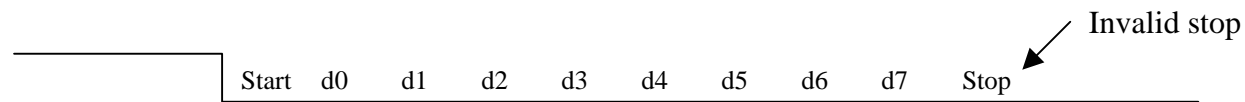
Since the receiver re-synchronises (starts the internal timing circuit) to the falling edge of each start bit, the timing of the transmitter and receiver only need to be the same to a tolerance of  $\frac{1}{2}$  a bit period in every 10 bit periods. This 5% accuracy is really no issue to achieve in any digital system.

In common with many UART solutions, these macros expect that a timing reference be provided in the form of an enable signal ('en\_16\_x\_baud'). which is applied at 16 times the bit rate

# UART Operation - Break Condition

The normal status of the serial line is active HIGH. In this way a new start bit is identified by its falling edge.

Under the break condition, a transmitter will continuously force a low level onto the line (possibly due to no power). Although the receiver will detect this as a start bit followed by all zero data, the stop bit will not be valid and this incorrect data will be discarded.



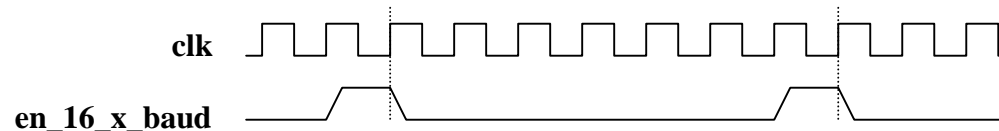
The receiver will then wait until the line returns high and will only resynchronise at the next falling edge associated with a start bit.



The transmitter macro will not naturally transmit a break condition. The receiver macro however does understand this situation and will operate as described above.

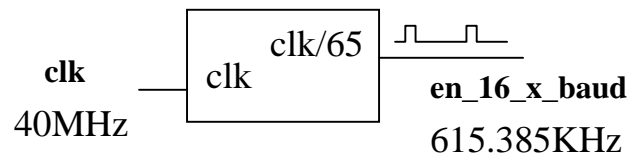
# BAUD Rate Timing

The macros derive the transmission and receive timing from a reference signal 'en\_16\_x\_baud'. As the name suggests, this signal should be applied to the macro at a rate which is 16× the desired bit rate.



Since the signal is used as a clock enable within the macros, it should be provided synchronous to the clock and have a pulse duration of one clock cycle only (unless the maximum communication rate of  $\text{clk}/16$  is required).

**Example** - The BAUD rate is required to be 38400Hz and the available system clock is 40MHz. This can be achieved by division of the clock  $40,000,000 / (16 \times 38,400) = 65.1$ . Although we can not provide pulses at exactly 65.1, the nearest integer of 65 is well in excess of the required tolerance (equivalent baud rate of 38461Hz which is just 0.15% high). Anything within 1% is really going to work as it allows for inaccurate clock rates and really poor switching on the serial lines.



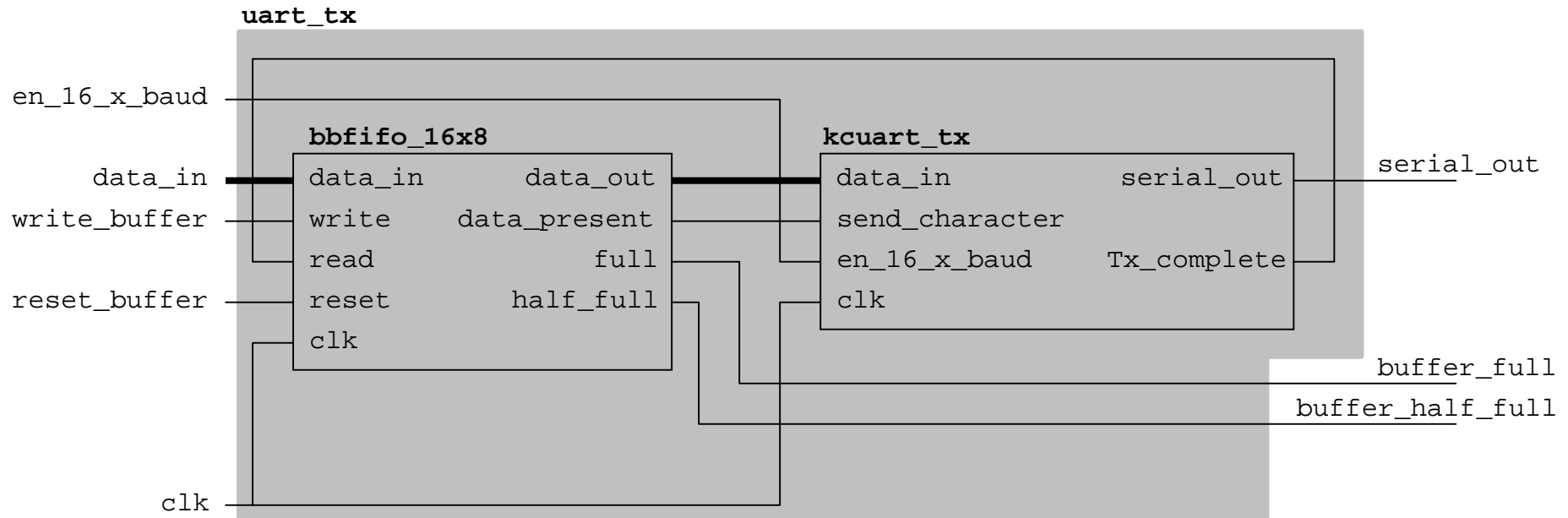
**Note** - Very efficient pulse generators can be formed using SRL16E components, and such techniques should be considered when the clock division factor and silicon utilisation is critical.

```
signal baud_count      : integer range 0 to 64 :=0;
...
baud_timer: process(clk)
begin
    if clk'event and clk='1' then
        if baud_count=64 then
            baud_count <= 0;
            en_16_x_baud <= '1';
        else
            baud_count <= baud_count + 1;
            en_16_x_baud <= '0';
        end if;
    end if;
end process baud_timer;
```



# The UART\_Tx Macro

The UART transmitter is provided formed by a set of three VHDL files. The top level file 'uart\_tx.vhd' is used to combine the FIFO buffer 'bbfifo\_16x8.vhd' and the constant(k) compact UART transmitter 'kcuart\_tx.vhd' modules.



## VHDL component data

```

component uart_tx
  Port (
    data_in : in std_logic_vector(7 downto 0);
    write_buffer : in std_logic;
    reset_buffer : in std_logic;
    en_16_x_baud : in std_logic;
    serial_out : out std_logic;
    buffer_full : out std_logic;
    buffer_half_full : out std_logic;
    clk : in std_logic);
end component;
```



# UART\_Tx Signals

**data\_in** - The parallel Byte (8-bit) data to be transmitted serially. The data will be captured by the FIFO buffer on a rising edge of the 'clk' during which 'write\_buffer' is active.

**write\_buffer** - An active HIGH indicates that the data currently being applied to the 'data\_in' port is to be written to the internal buffer on the next rising edge of 'clk'. A write operation will take place on every rising clock edge on which this signal is active. Hence this signal should be pulsed HIGH for one clock cycle only, unless new data is applied to 'data\_in' every clock cycle for a 'burst write'. The FIFO will ignore data should it become full.

**reset\_buffer** - An active HIGH input causes the 16-byte internal buffer to be reset and hence all data currently in the buffer to be lost. Operation of this signal during serial data transmission will potentially result in corrupted data.

**en\_16\_x\_baud** - Provides the timing reference for the serial transmission. This input should be pulsed HIGH for one clock cycle duration only and at a rate which is 16 times (or approximately 16 times) the rate at which the serial data transmission is to take place. Alternatively, this signal may be set continuously HIGH such that serial data transmission takes place at the maximum rate of clk/16 bits per second.

**serial\_out** - This is the serial data conforming to 1 start bit, 8 data bits (LSB first), and 1 stop bit. In accordance with normal UART operation, this signal is HIGH in the idle (no data to transmit) condition. Serial data transmission commences as soon as there is data in the buffer and continues without interruption (start bit immediately follows stop bit) until the buffer is empty.

**buffer\_full** - When the 16-byte FIFO buffer is full, this output becomes active HIGH. The host system should not attempt to write any new data until the serial transmission has been able to create a space (indicated by 'buffer\_full' returning low). Any attempt to write data will mean that the new data is ignored.

**buffer\_half\_full** - When the 16-byte FIFO buffer holds eight or more bytes of data waiting to be transmitted, this output becomes active HIGH. This is a useful indication to the host system that the FIFO buffer is approaching a full condition, and that it would be wise to reduce the rate at which new data is being written to the macro.

**clk** - Used by all synchronous elements of the macro, this signal should be provided via one of the global low-skew clock networks and all other signals should be applied and read synchronously to this clock.

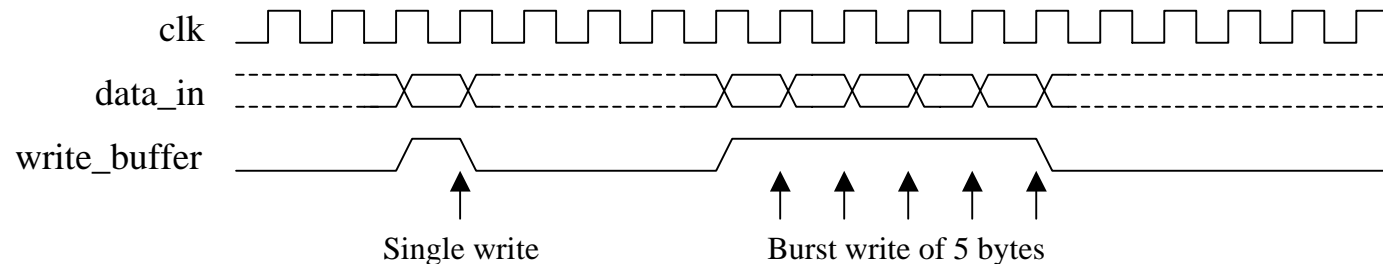


# Tx Buffer Operation

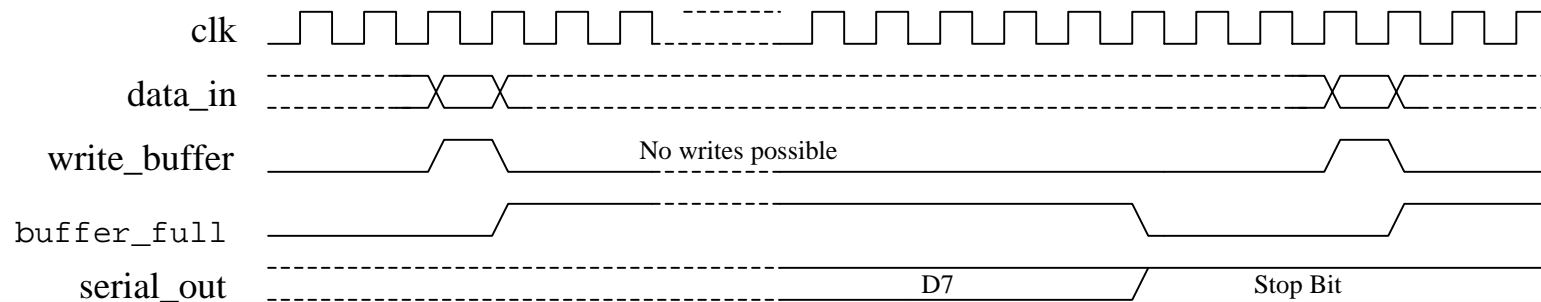
If required, the 'bbfifo\_16x8.vhd' can be used to provide a synchronous 16-byte FIFO buffer using just 8 'slices'. As part of the 'uart\_tx' macro, the output side of the FIFO is under the control of the 'kcuart\_tx' transmitter.

'reset\_buffer' can be used to discard the current contents of the buffer and obviously needs to be used with care so as not to destroy important information. It is also possible that any serial data transmission in progress will be corrupted. In most cases this signal will not be used as the FIFO will be fully reset following configuration of the device.

The FIFO buffer is used to accept byte data for transmission when written to the macro. The buffer is automatically read by the 'kcuart\_tx' circuit to pass the data to the serial line. Data is written to the buffer on the rising edge of clock when 'write\_buffer' signal is active. Data can be written in isolation, or in a burst of several bytes.

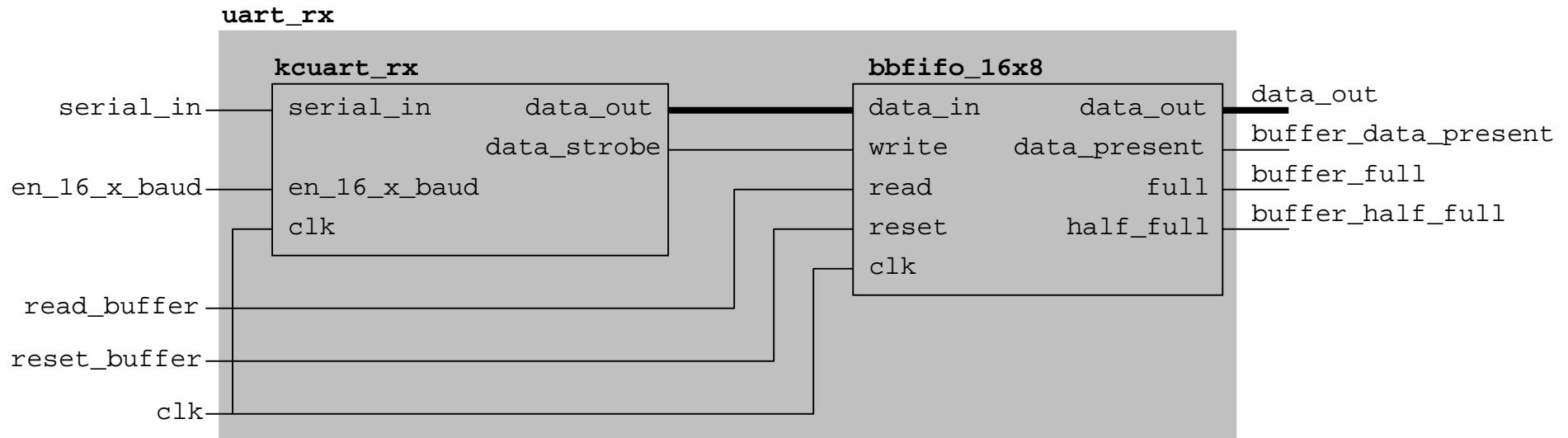


When the FIFO buffer becomes full, the 'buffer\_full' signal will be asserted. This signal will remain asserted until the MSB of the currently being transmitted data is complete (ie once the next stop bit is being transmitted). No data can be written to the buffer when it is full.



# The UART\_Rx Macro

The UART receiver is provided formed by a set of three VHDL files. The top level file 'uart\_rx.vhd' is used to combine the constant(k) compact UART receiver 'kcuart\_rx.vhd' and FIFO buffer 'bbfifo\_16x8.vhd' modules.



## VHDL component data

```

component uart_rx
  Port (
    serial_in : in std_logic;
    data_out  : out std_logic_vector(7 downto 0);
    read_buffer : in std_logic;
    reset_buffer : in std_logic;
    en_16_x_baud : in std_logic;
    buffer_data_present : out std_logic;
    buffer_full : out std_logic;
    buffer_half_full : out std_logic;
    clk : in std_logic);
end component;
  
```

# Rx Operation and Signals

**serial\_in** - This is the serial data conforming to 1 start bit, 8 data bits (LSB first), and 1 stop bit. In accordance with normal UART operation this signal is HIGH in the idle condition. The falling edge associated with a start bit is used to identify the beginning of a serial transmission and the en\_16\_x\_baud is used to determine the timing of the transfer. Once a complete serial transfer has been received with a valid stop bit, it is automatically written to the FIFO buffer (unless the buffer is full).

**data\_out** - The parallel Byte (8-bit) data which has been received. This data is valid when 'buffer\_data\_present' is active.

**read\_buffer** - An active HIGH input indicates that the data provided at the 'data\_out' port has been read (or will be read on the next rising edge of 'clk') and that the FIFO should make the next available data available. The 'read\_buffer' input may be active for consecutive clock cycles to perform a 'burst' of data. Any attempt to read data when 'buffer\_data\_present' is inactive will have no effect, but this illegal case should be avoided when possible.

**reset\_buffer** - An active HIGH input causes the 16-byte internal buffer to be reset and hence all data currently in the buffer to be lost.

**en\_16\_x\_baud** - Provides the timing reference for the serial transmission. This input should be pulsed HIGH for one clock cycle duration only and at a rate which is 16 times (or approximately 16 times) the rate at which the serial data transmission is taking place. Alternatively, this signal may be set continuously HIGH such that serial data is received at the maximum rate of clk/16 bits per second.

**buffer\_data\_present** - When the internal buffer contains one or more bytes of received data this signal will become active HIGH and valid data will be available at the 'data\_out' port.

**buffer\_full** - When the 16-byte FIFO buffer is full, this output becomes active HIGH. The host system should rapidly respond to this condition by reading some data from the buffer so that further serial data is not lost.

**buffer\_half\_full** - When the 16-byte FIFO buffer holds eight or more bytes of data waiting to be read, this output becomes active HIGH. This is a useful indication to the host system that the FIFO buffer is approaching a full condition, and that it would be wise to read some data in the very near future.

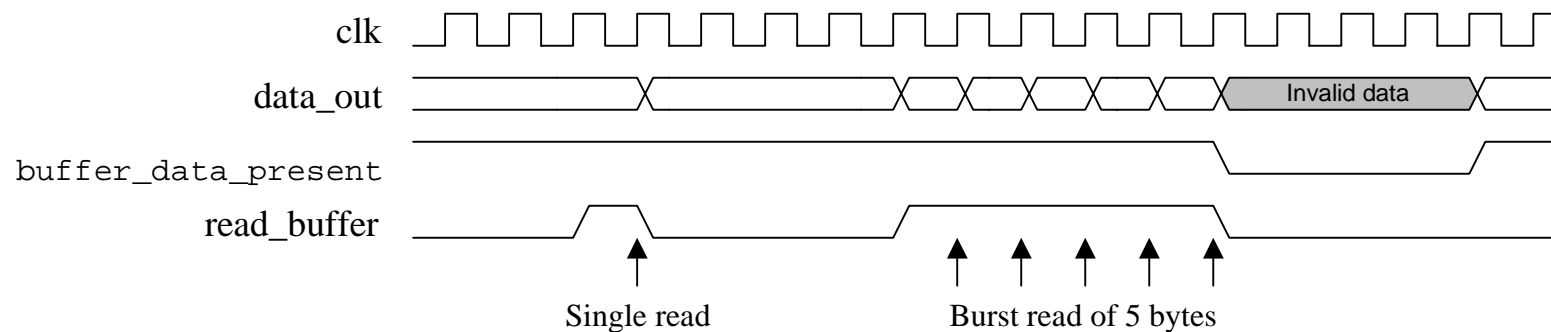
**clk** - Used by all synchronous elements of the macro, this signal should be provided via one of the global low-skew clock networks and all other signals (except 'serial\_in' for obvious reasons) should be applied and read synchronously to this clock.

# Rx Buffer Operation

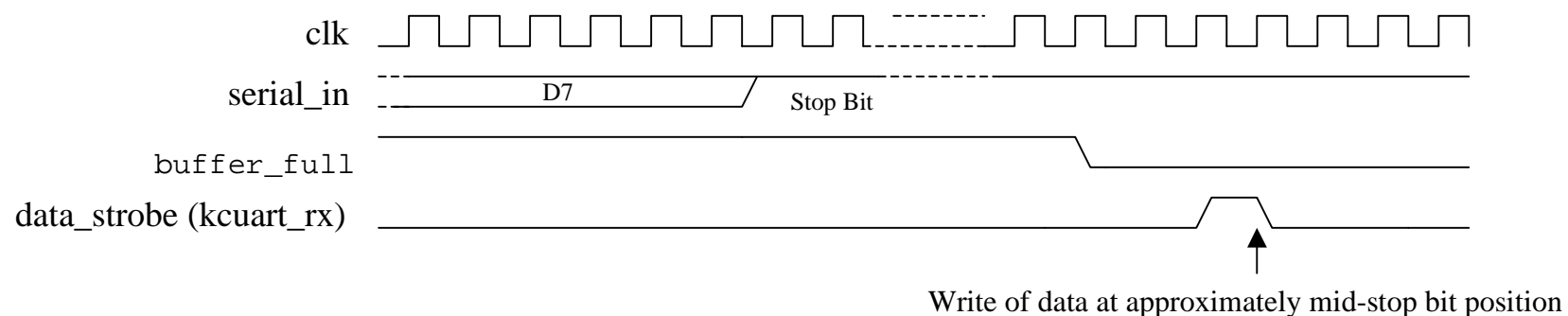
If required, the 'bbfifo\_16x8.vhd' can be used to provide a synchronous 16-byte FIFO buffer using just 8 'slices'. As part of the 'uart\_rx' macro, the input side of the FIFO is under the control of the 'kcuart\_rx' transmitter.

'reset\_buffer' can be used to discard the current contents of the buffer and obviously needs to be used with care so as not to destroy important information. In most cases this signal will not be used as the FIFO will be fully reset following configuration of the device.

When data is present (buffer\_data\_present=1), individual or burst reads can be made from the FIFO buffer.



The FIFO buffer is automatically written by the 'kcuart\_rx' circuit as each valid serially transmitted 'character' is captured by the receiver. Data is written to the buffer as soon as the stop bit has been confirmed as being high, but only if the buffer is not full.



It can be seen above that when the FIFO buffer becomes full, it doesn't actually prevent the next 'character' from being received. Therefore, the host system has the time associated with ten serial bits to read at least one byte from the FIFO before data is actually missed.