XAPP223 (v1.0) January 31, 2001

200 MHz UART with Internal 16-Byte Buffer

Author: Ken Chapman

Summary

This application note describes highly optimized UART transmitter and receiver macros for Xilinx Virtex™, Virtex-E, and Spartan™-II devices. The UART_TX and UART_RX macros not only communicate with each other, but they are also fully compatible with the standard Universal Asynchronous Receiver Transmitter (UART) communication protocols used for connecting to devices, such as PCs or microcontrollers.

General Description

The UART_TX and UART_RX macros provide the functionality of a simple UART transmitter and receiver, each with the following fixed characteristics:

- 1 start bit
- 8 serially transmitted data bits, received least significant bit (LSB) first
- 1 stop bit

Each macro also contains an embedded 16-byte first-in first-out (FIFO) buffer, using a total of only 15 Configurable Logic Blocks (CLBs). See Figure 1.

The macros are implemented using SMART_IP for absolute predictability over shape and size and for providing very high levels of performance. Indeed, the Virtex-E 8-speed grades ensure operation at clock rates exceeding 200 MHz. This high-hevel performance combined with the internal data buffers results in asynchronous data transfers in excess of 1 Mbyte per second (Mb/s) with these macros.

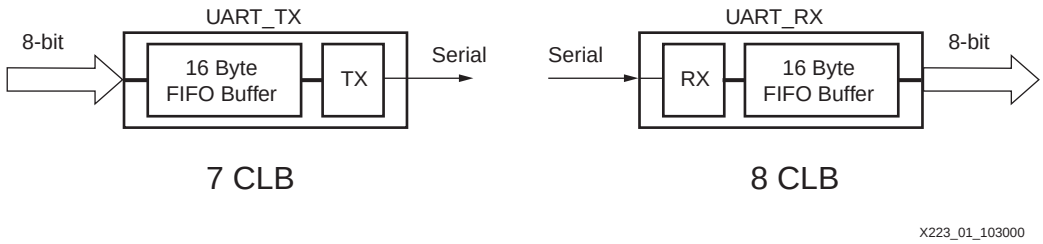

Figure 1: **UART Transmitter and Receiver Macros**

Detailed Description

UART_TX Macro

The UART transmitter is provided as a single EDIF netlist, which can be instantiated into a design, as shown in Figure 2.

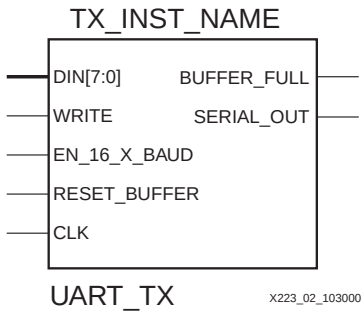

Figure 2: **UART_TX Macro**

Schematic Symbol

From the Foundation schematic, select “Create Macro Symbol from Netlist."

VHDL Component Data

Using Synthesis Tools, select the appropriate pin and component names.

```
component uart_tx is
  port (            din : in STD_LOGIC_VECTOR (7 downto 0);
                  write : in  STD_LOGIC;
           reset_buffer : in  STD_LOGIC;
           en_16_x_baud : in  STD_LOGIC;
                    clk : in  STD_LOGIC;
             serial_out : out STD_LOGIC;
            buffer_full : out STD_LOGIC);
  end component;
```

UART_RX Macro

The UART receiver is provided as a single EDIF netlist, which can be instantiated into a design, as shown in Figure 3.

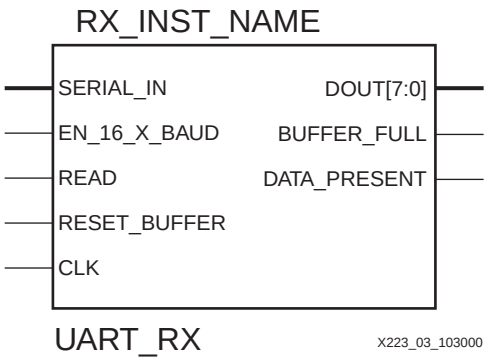

Figure 3: **UART_RX Macro**

Schematic Symbol

From the Foundation Schematic, select “Create Macro Symbol From Netlist.”

VHDL Component Data

Using Synthesis Tools, select the appropriate pin and component names.

```
component uart_rx is
  port (      serial_in : in  STD_LOGIC;
                   read : in  STD_LOGIC;
             reset_buffer : in  STD_LOGIC;
             en_16_x_baud : in  STD_LOGIC;
                      clk : in  STD_LOGIC;
                     dout : out STD_LOGIC_VECTOR (7 downto 0);
             data_present : out STD_LOGIC;
              buffer_full : out STD_LOGIC);
  end component;
```

Common Signals

The signals shown in Table 1 are provided on both the transmitter and receiver macros.

Table 1: **Signals Common to Both Transmitter and Receiver**

Signal	Direction	Description
CLK	Input	This signal is the system clock that is used by all elements forming the UART and buffer functionality. This signal should be provided using one of the global low-skew clock networks, and all other signals should be applied and read synchronously to this clock. (Note: SERIAL_IN on RX is exempt in an asynchronous UART).
EN_16_X_BAUD	Input	This input provides the basic timing for a serial transmission. This input should be pulsed HIGH for one clock cycle duration only and at a rate 16 times (or approximately 16 times) the rate at which the serial data transmission takes place. Alternatively, this signal can be set continuously HIGH, such that serial data transmission takes place at CLK/16 bits per second.
RESET_BUFFER	Input	An active HIGH input causes the 16-byte internal buffer to be reset; hence, all data currently in the buffer is lost. Operation of this signal during serial transmission potentially results in corrupted data. The buffers are initialized to an empty state following power-up. Therefore, this signal is not used in most cases.
BUFFER_FULL	Output	This signal becomes active HIGH when the internal 16-byte buffer is full. For the transmitter, it means that the host system should not apply (WRITE) any new data until the serial transmission is able to create a space (BUFFER_FULL returns LOW). Any attempt to write data simply means that the new data is ignored. At the receiver, it means that 16 received bytes are waiting to be read. This signal should be an indication to the host system to rapidly read data from the buffer, because all subsequent serial data will be lost.

UART_TX Specific Signal Descriptions

The signals shown in Table 2 are specific to the transmitter module. In normal operation, the data bytes are written rapidly into the internal buffer; the transmitter then delivers them to the serial output at a relatively slow rate.

Table 2: **Transmitter Signals**

Signal	Direction	Description
DIN[7:0]	Input	This signal is the parallel byte (8-bit) data to be transmitted serially. This is the data input to the internal buffer, which should be stable during an active WRITE clock cycle.
WRITE	Input	An active HIGH input indicates that the data currently being applied to the DIN[7:0] port is written to the internal buffer on the next rising clock edge. Note that write operations take place on every rising edge when this signal is active HIGH. Hence, this signal should be pulsed HIGH for one cycle only, unless new data is applied to DIN[7:0] every clock cycle for a “burst write.” Note: The data is not stored if the BUFFER_FULL signal is active HIGH.
SERIAL_OUT	Output	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. 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.

UART_RX Specific Signal Descriptions

The signals shown in Table 3 are specific to the receiver module. In normal operation, the serial data is received at a relatively slow rate and collected in the internal FIFO buffer. The resulting data bytes are then read rapidly from the internal buffer.

Table 3: **Receiver Signals**

Signal	Direction	Description
DOUT[7:0]	Output	This port provides the parallel byte (8-bit) data that has been received. This data is valid when DATA_PRESENT is active HIGH.
DATA_PRESENT	Output	This signal is active HIGH when the internal buffer contains one or more bytes of received data. It also signifies that DOUT[7:0] is valid data.
READ	Input	An active HIGH input indicates that the data currently being provided at the DOUT[7:0] port has been read (or will be read on the next rising clock edge) and that the next available data can be made available. The READ input can be applied for consecutive clock cycles to perform a “burst” of data. An attempt to READ with DATA_PRESENT inactive has no effect; however, this illegal case should be avoided whenever possible.

Table 3: **Receiver Signals *(Continued)***

SERIAL_IN	Input	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 written to the internal buffer, provided that BUFFER_FULL is not active.

UART Operation

Although an asynchronous receiver and transmitter are not synchronized, the UART_TX and UART_RX both use a timing reference that is of adequate 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) 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. See Figure 4.

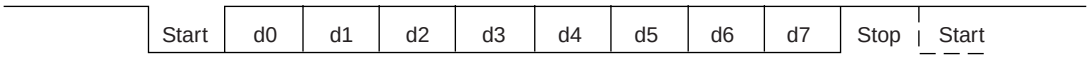

Figure 4: **UART Operation - Transmitter**

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 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. This helps confirm correct operation. See Figure 5.

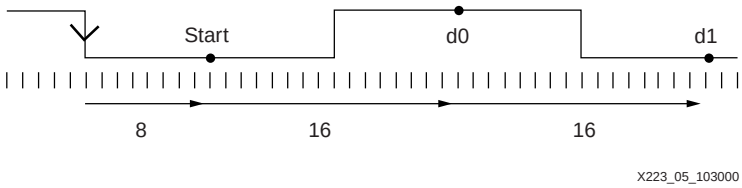

Figure 5: **UART Operation - Receiver**

Since the receiver resynchronizes (starts the internal timing circuit) to the falling edge of each start bit, the transmitter and receiver timing must be the same to a accuracy of only one-half of a bit period every 10-bit periods. This 5-percent tolerance is usually easy to achieve in digital systems.

In common with many UART solutions, these macros expect that a timing reference be provided in the form of an enable signal applied at 16 times the bit rate (EN_16_X_BAUD).

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 continuously forces a low level onto the line (possibly due to no power). Although the receiver detects this as a start bit followed by all zero data, the stop bit is not valid and, therefore, this incorrect data is discarded. See Figure 6.

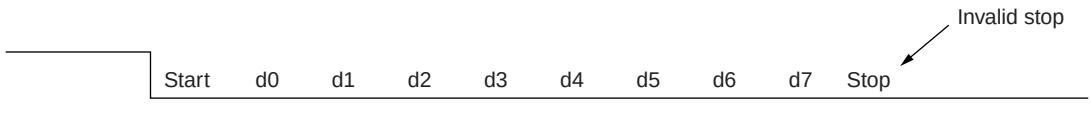

Figure 6: **Break Condition - Transmitter**

The receiver then waits until the line returns HIGH and resynchronizes only at the next falling edge associated with a start bit. See Figure 7.

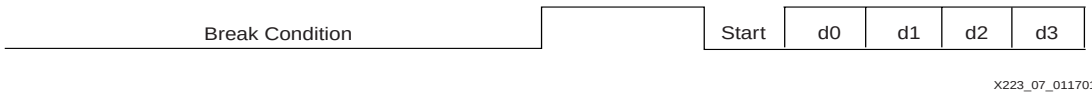

Figure 7: **Break Condition - Receiver**

The transmitter macro does not naturally transmit a break condition. The receiver macro, however, does understand this situation and operates as shown in Figure 7.

Buffer Operation

The buffer in each macro performs a FIFO operation on the byte data to a depth of 16 bytes.

RESET_BUFFER can be used to discard the current contents of the buffer and, therefore, it must be used with caution, so that data is not lost. In most cases, this signal is not used.

The UART_TX buffer is used to accept byte data for transmission when written to the macro. The buffer is automatically read by the transmission circuit to pass the data to the serial line. When the WRITE signal is active, data is written to the buffer on the rising edge of clock. Data can be written in isolation, or in a burst of several bytes. See Figure 8.

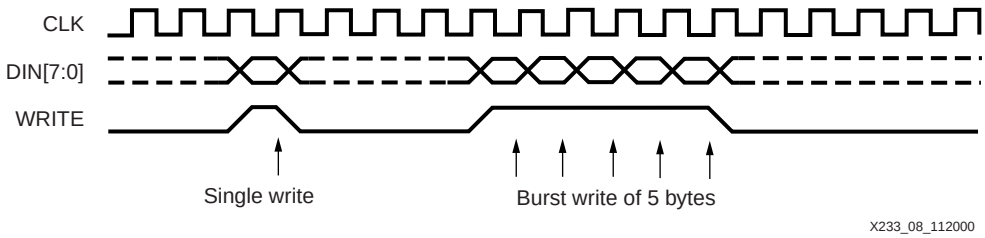

Figure 8: **UART_TX Buffer Operation**

When the UART_TX buffer is full, the BUFFER_FULL signal is asserted by the macro. This signal remains asserted until the MSB of the currently transmitted data is complete (i.e., once the next stop bit is being transmitted). No data can be written to the buffer when it is full. See Figure 9.

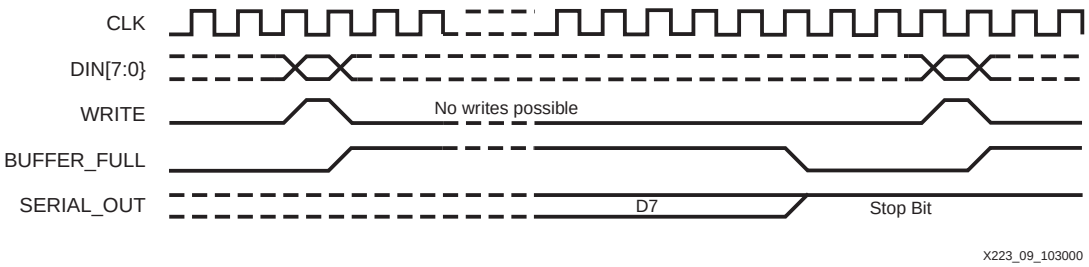

Figure 9: **UART_TX Buffer Full**

The UART_RX buffer is used to store data received automatically from the serial line until it can be read. As soon as there is data available, the DATA_PRESENT signal is asserted on the macro. The next data to be read is provided at the DOUT[7:0] port. The application of an active HIGH READ signal indicates that the current data has been read and the next data will be made available. The read operation can be performed in isolation or as a burst of several reads (as long as there is adequate data present). The application of READ when DATA_PRESENT is not active has no effect, but clearly the data is not valid. See Figure 10.

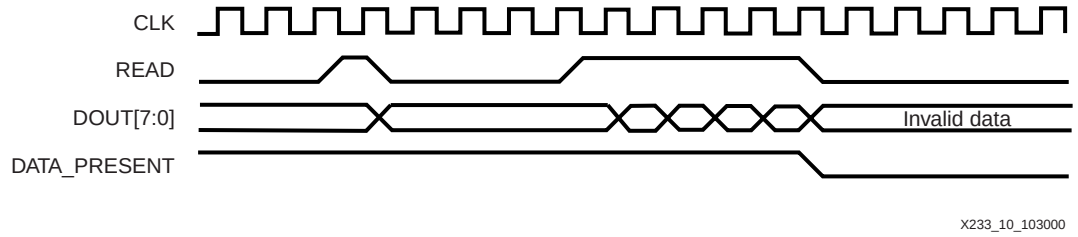

Figure 10: **UART_RX Buffer Operation**

When the UART_RX buffer becomes full, the BUFFER_FULL signal is asserted by the macro. Although the host system rapidly executes some READ cycles to clear space in the buffer, serial data continues to be collected. Only when the next valid stop bit following the assertion of BUFFER_FULL is encountered is data actually lost. Therefore, at least one READ cycle should be performed before the time taken to transmit one data byte serially has expired. See Figure 11.

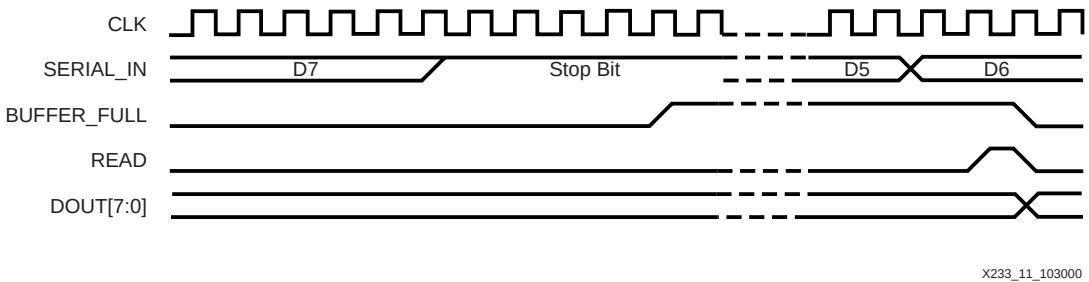

Figure 11: **UART_RX Buffer Full**

Standard UART Applications

The UART_TX and UART_RX macros can be used to enable a Xilinx device to communicate with other devices conforming to standard UART protocols, such as PCs and microcontrollers. If full RS232 signaling is employed, then an external device such as the MAXIM MAX220 (multi-channel RS232 drivers/receivers) should be used. In cases where a Xilinx device is performiing microcontroller communication on the same printed circuit board (PCB), then a direct low voltage transistor transistor logic (LVTTL) should be more than adequate. However, in this case, it is required that the external device must conform to the fixed protocol of the macros (1 start bit, 8 data bits, no parity, and 1 stop bit). See Figure 12.

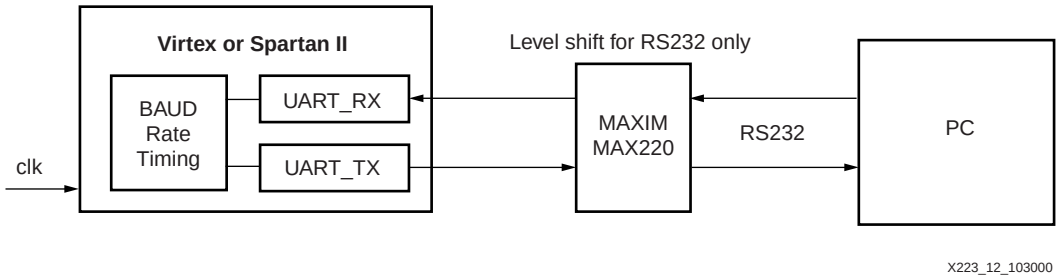

Figure 12: **Xilinx Device Communication Using Standard UART Protocols**

In this situation, it can also be expected that the serial bit rate is relatively slow and conforms to one of the RS232 BAUD rates, such as 4800, 9600, 19200, 38400, etc. This requires that the timing be provided to the macros within the Xilinx device within an acceptable tolerance. See Figure 13.

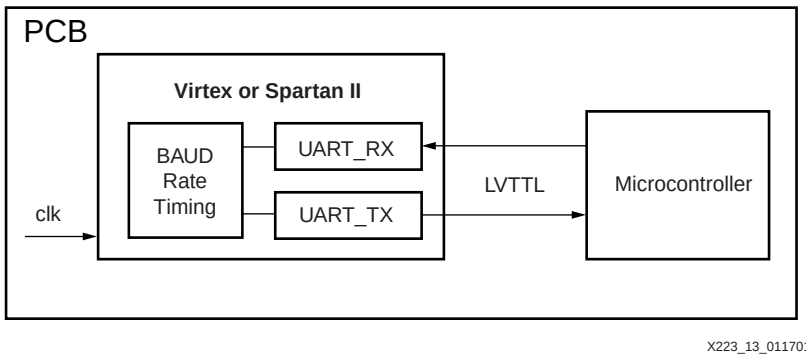

Figure 13: **Xilinx Device Communication to a Microcontroller**

BAUD Rate Timing

The macros derive the transmission and receive timing from the reference signal, EN_16_X_BAUD. This signal should be applied to the macro at a rate 16 times faster than the desired bit rate. Figure 14.

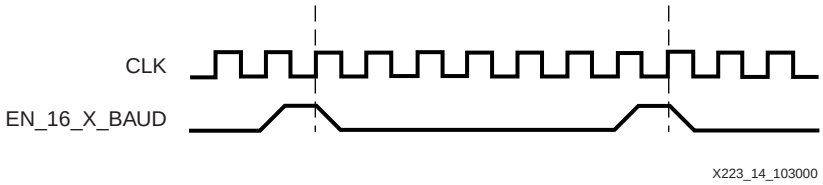

Figure 14: **Baud Rate Timing**

Since the signal is used as a clock enable within the macros, it should be provided synchronously to the clock and have a pulse duration of one clock cycle only.

Example

The required BAUD rate is 9600 Hz and the available system clock is 40 MHz.

EN_16_X_BAUD = 16 × 9600 Hz = 153,600 Hz

This can be achieved by division of the clock 40 MHz/153600 = 260.41. See Figure 15.

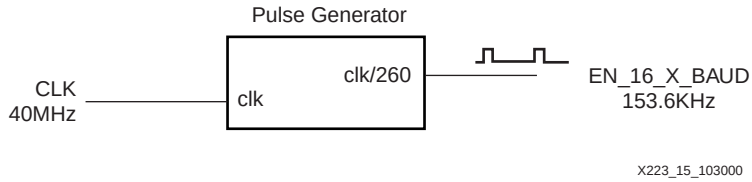

Figure 15: **Timing Example**

Although pulses cannot be provided at exactly 260.41, the nearest integer of 260 is well in excess of the required tolerance (equivalent baud rate of 9615 Hz which is just 0.16% high). Anything within 1 percent works since it allows for inaccurate clock rates and poor switching on the serial lines.

Timing Pulse Generators

Pulse generators are easy to implement and require only one timing generator to service the transmitter and receiver macros (assuming both are at the same baud rate).

A simple way to achieve a division for the example of CLK/260 is with a 9-bit counter. This counter has the ability to count to 511, but an AND gate detection of the count value 259 is used to provide the output pulse and synchronously reset the counter. See Figure 16.

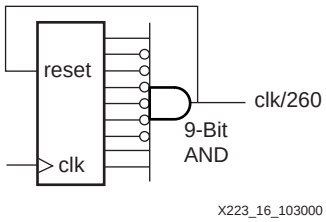

Figure 16: **Timing Pulse Generator**

Although the counter method is straightforward, it does require six “slices” of logic (three CLBs) in this example. If cost is critical, then every optimization must be considered. The circuit in Figure 17 also generates pulses at CLK/260, but does so in just 1 CLB. The SRL16E shift registers of Virtex and Spartan-II devices are used as highly efficient one-hot state machine counters.

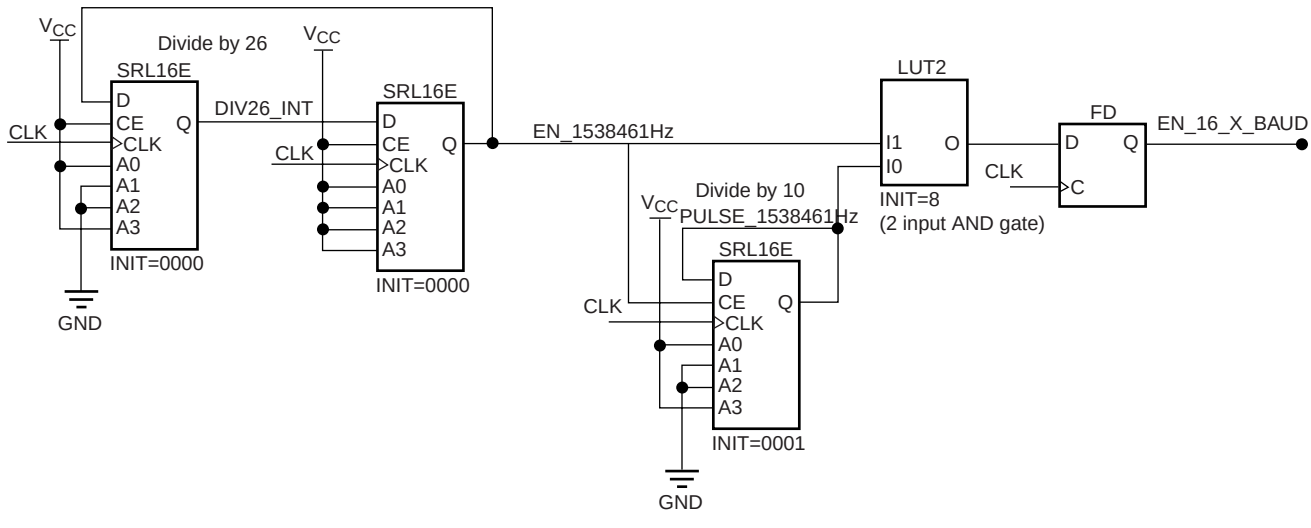

Figure 17: **Pulse Generator Circuit**

Embedded Microcontroller

The ability to implement efficient serial communications inside a Virtex or Spartan-II device is useful. However, communications carried out on such serial links are often made using higher level protocols, consisting of the ASCII character set with control characters, such as LF and CR. The particular application can also use special character sequences to identify where data packets are contained. Although hardware state machines could be created to deal with these cases, it is likely to be a difficult design task that would result in large silicon requirements.

A small 8-bit microcontroller called Constant Coded Programmable State Machine (KCPSM) has been developed and is ideal for performing such tasks. See Figure 18. The internal buffers of the UART macros enable the processor to work efficiently on the data without servicing interrupts all the time (interrupts can be left to look for BUFFER_FULL conditions). At 35 CLBs, this processor is very small; hence, the whole serial communications controller solution is possible in the smallest Spartan-II device or virtually free in Virtex devices.

More details on the 8-bit microcontroller, KCPSM, can be found in the Xilinx Application Note, XAPP213 at: **http://www.xilinx.com/xapp/xapp213.pdf**

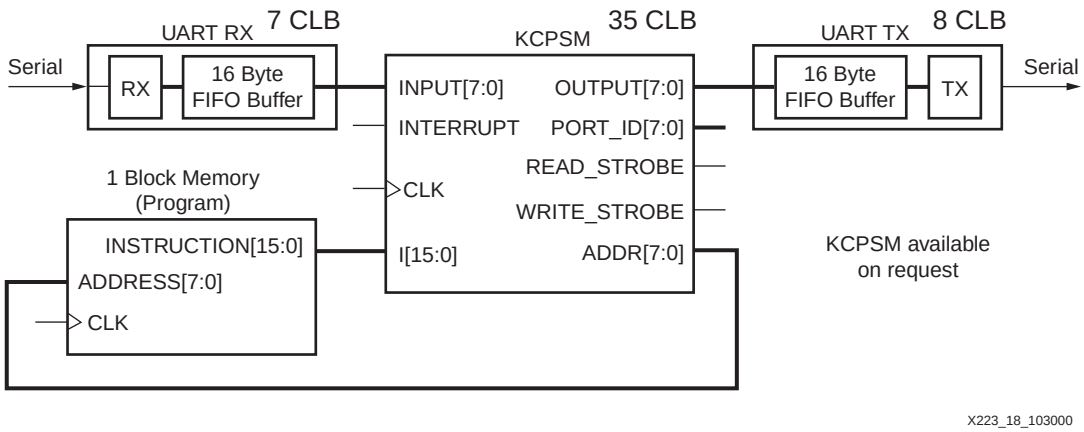

Figure 18: **Embedded Microcontroller**

UART FastLINK

The UART_TX and UART_RX macros can operate at clock rates in excess of 200 MHz. This performance can be exploited to provide fast asynchronous serial communications between Xilinx devices. Since there is no reason to conform to any of the standard BAUD rates, the EN_16_X_BAUD can be connected permanently HIGH, so that the BAUD rate is the clock/16. See Figure 19. For example, a 160-MHz clock results in a 10-Mbit/s serial transfer; hence, a data rate of 1 Mbyte/s (10-bit periods per 8-bit data transfer).

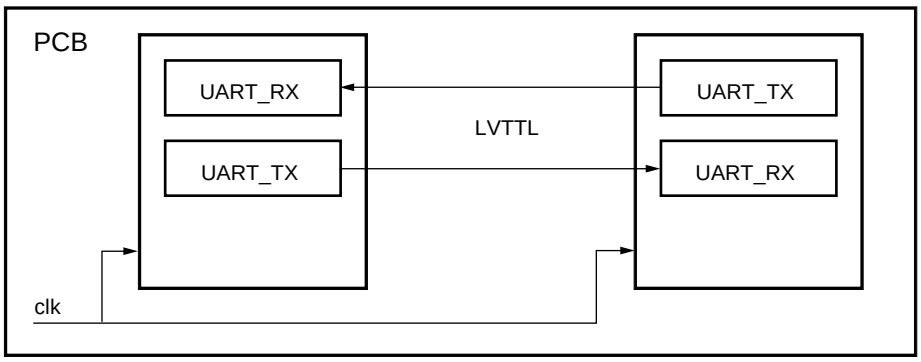

Figure 19: **UART FastLINK**

Of greater interest is the use of fast UART serial links between Xilinx devices on different boards (or equipment boxes). Use of built-in I/O standards, such as low voltage differential signaling (LVDS), avoids noise problems. The asynchronous nature of UART communication avoids issues of clock skew and signal delay between components. See Figure 20.

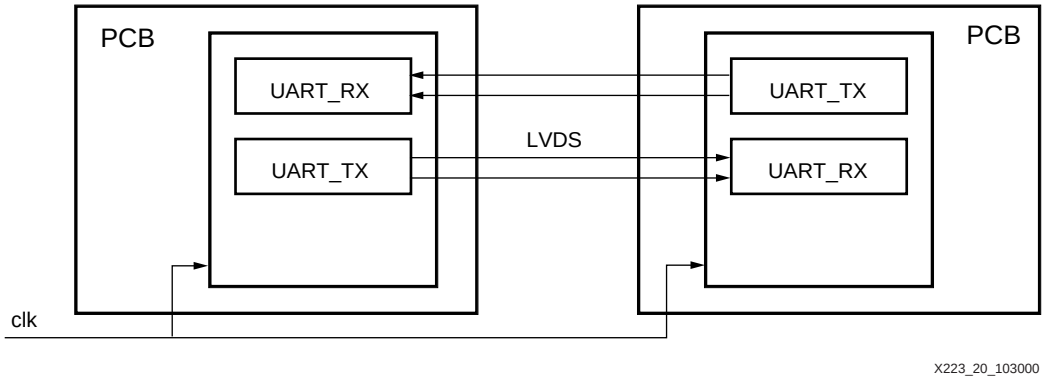

Figure 20: **Fast UART Serial Links Between Xilinx Devices**

Size and Performance

The macros are provided as Relationally Placed Macros (RPM) of the shape shown in Figure 21.

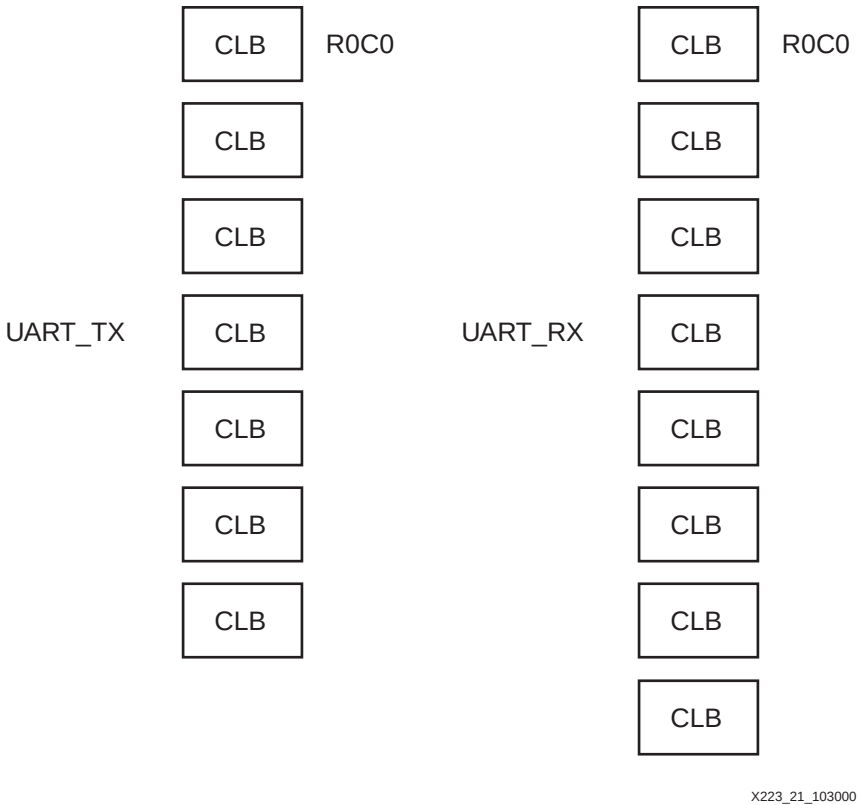

Figure 21: **Transceiver and Receiver Relationally Placed Macros**

Each macro is capable of clock operation in excess of 200 MHz, which is suitable for 10 Mbit/s asynchronous communication. In most cases, the baud rate is significantly lower, making performance of little concern to the user. The ability of these macros to operate so fast, however, is of use when applying time specifications to the rest of your system. This avoids the requirement for special (slow) time specifications that must be applied specifically to these macros.

The most interesting aspect of size and performance is perhaps that these combined 15 CLBs contain an equivalent of 10,000 ASIC gates. This means that a device, such as the Spartan XC2S50, could implement 25 UART transmitter and receiver pairs with an equivalent gate count of 250,000 gates, representing a great value for little cost.

Downloading the Macros

The UART_TX and UART_RX macros are available by downloading **XAPP223.zip**. The macros are in the form of EDIF (.edn) files which can be used within any Virtex or Spartan-II design. The macros should be treated as a 'black box' in the design. Other information about the macros is given in the Read Me file.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
01/31/01	1.0	Initial Xilinx release