Communicating through Virtual UARTs

This application note describes how Virtual UARTs work, and presents sample code to demonstrate the principles. A virtual UART is a device which allows an embedded processor (target) and a development computer (HOST) to communicate through a memory socket. There are several different methods of accomplishing this, each with different strengths and weaknesses. The following sections discuss each method.

READ/WRITE Virtual UARTs

The first type of Virtual UART is basically a standard UART that is mapped into your target's memory space. Your target program interacts with this type UART very much like any other UART. The primary difference is that it occupies some of your CODE space. This is the type of Virtual UART that is included with UniROM as standard equipment. UniROM's built-in Virtual UART is called VCOM. VCOM uses 4 bytes of shared memory to facilitate communications. The target sends a character to the HOST by writing the character to a specific location in its EPROM or FLASH memory space. It then sets a bit in another location to inform UniROM that the TX REGISTER is full. UniROM then echoes that character to its serial port and clears the FULL flag. Likewise, when UniROM receives a character from the HOST, it writes that data into a specified memory location, sets a RX FULL flag and generates an interrupt to the target. The target can poll the RX FULL bit to determine when data is available, or it can hook an interrupt routine. The target reads the received byte and then clears the RX FULL flag to indicate that it has been received. This is illustrated in the code fragments below:

```
// assume tx_full_flag, rx_full_flag, rx_buff and tx_buff are unsigned char pointers that have been initialized
// to point to the appropriate memory mapped locations for the VCOM.

void send_char(unsigned char tx_data)
{while (*tx_full_flag)                  // wait for the transmit buffer to empty
      continue;
  *tx_buff = txdata;                    // fill the TX buffer
  *tx_full_flag = 0x01;                 // set the TX FULL flag
}

unsigned char get_char(void)
{ unsigned char ch;
  while ( *rx_full_flag  == 0)          // wait for a character to arrive
      continue;
  ch = *rx_buffer;                      // fetch the character
  *rx_full_flag = 0;                    // clear the RX FULL flag
  return(ch);
}
```

As you can see this operates very much like any other standard UART. The only difference is that the target software is responsible for setting the TX FULL flag and clearing the RX FULL flag. This type of Virtual UART requires that the target is capable of writing to the EPROM or FLASH space the emulator is plugged into.

READ ONLY Virtual UARTs

Often, targets are not capable of writing to their EPROM or FLASH code space. This may be due to uni-directional data buffers, incomplete chip select decoding or improper write cycle timing. To insure complete compatibility with ALL targets, we have developed a method that allows the target to "transmit" without using write cycles. This is accomplished by the target performing a special sequence of reads to specific memory locations to encode the data. We offer a URCOM option for UniROM which uses this method. In addition, we offer a stand-alone product called ROMCOM. ROMCOM supports 3 different encoding schemes as well as a traditional read/write mode. Some encoding methods are very memory efficient but require several target cycles to encode the data. Others use a lot of memory, but encode the data with a single target read cycle. RomCOM allows one to select the method that matches their requirements in any particular application.

 (972) 272-9392 email: sales@tech-tools.com

URCOM uses a 3 bit encoding method that encodes a transmit character in just 3 target read cycles and requires only 16 bytes of EPROM space for the UART footprint. We feel this is the best combination. Even slower embedded controllers can sustain 115Kbaud transfer rates. RomCOM also supports this method.

The following code fragments demonstrate how a target "SENDS" a character through a virtual UART using each of the encoding methods supported by URCOM or RomCOM.

// assume that tx_base is an unsigned char pointer to the start of Virtual UART address space
// and that status is an unsigned char pointer to the address of the status register in that space.

```
// 3 bit encoding  (requires 16bytes of memory and 3 target reads.  Compatible with URCOM/UniROM)
Void Send_char(unsigned char tx_data)
{  unsigned char dummy;
   while(*status & 0x01)                          // wait for the transmit register to empty
       continue;
  dummy = *(tx_base + (tx_data&0x07));            // encode d0-2
  tx_data = tx_data >> 3;
  dummy = *(tx_base + (tx_data&0x07));            // encode d3-5
  tx_data = tx_data >> 3;
  dummy = *(tx_base + (tx_data&0x03) + 8)         // encode d6-7 and set full flag
}

// 1bit encoding  (requires 4 bytes of memory and 10 target reads)
Void Send_char(unsigned char tx_data)
{  unsigned char dummy, i;
   while(*status & 0x01)                          // wait for the transmit register to empty
       continue;
   dummy = *tx_base;                              // send a start bit
   for(i=0;i<8;i++)                               // send each data bit
     { if(tx_data&0x01)
          dummy = *(tx_base+1);                   //  encode a '1'
       else
          dummy = *tx_base;                       //  encode a '0'
       tx_data >>= 1;
     }
   dummy = *(tx_base+1)                           // send stop bit
}

// 8 bit encoding   (requires 512 bytes of memory and 1 target read)
Void Send_char(unsigned char tx_data)
{  unsigned char dummy;
   while(*status & 0x01)                          // wait for the transmit register to empty
       continue;
   dummy = *(tx_base + tx_data);                  // encode d0-7
}
```

URCOM and RomCOM both include complete working code examples in C and some popular assembly languages.

All remote-style software debuggers provide a means of configuring the remote kernel to match your target UART's address, IRQ structure and register set. Since these debuggers must work with a variety of custom targets, they must allow for easy configuration for whatever UART your target may be using. Many debuggers offer one or more of these encoding methods as a standard installation option. Even if your debugger is not pre-configured to use one of these methods, it is extremely easy to configure it yourself, using the uart_get(), uart_put(), uart_init() and uart_status() routines we provide as examples.

 (972) 272-9392 email: sales@tech-tools.com