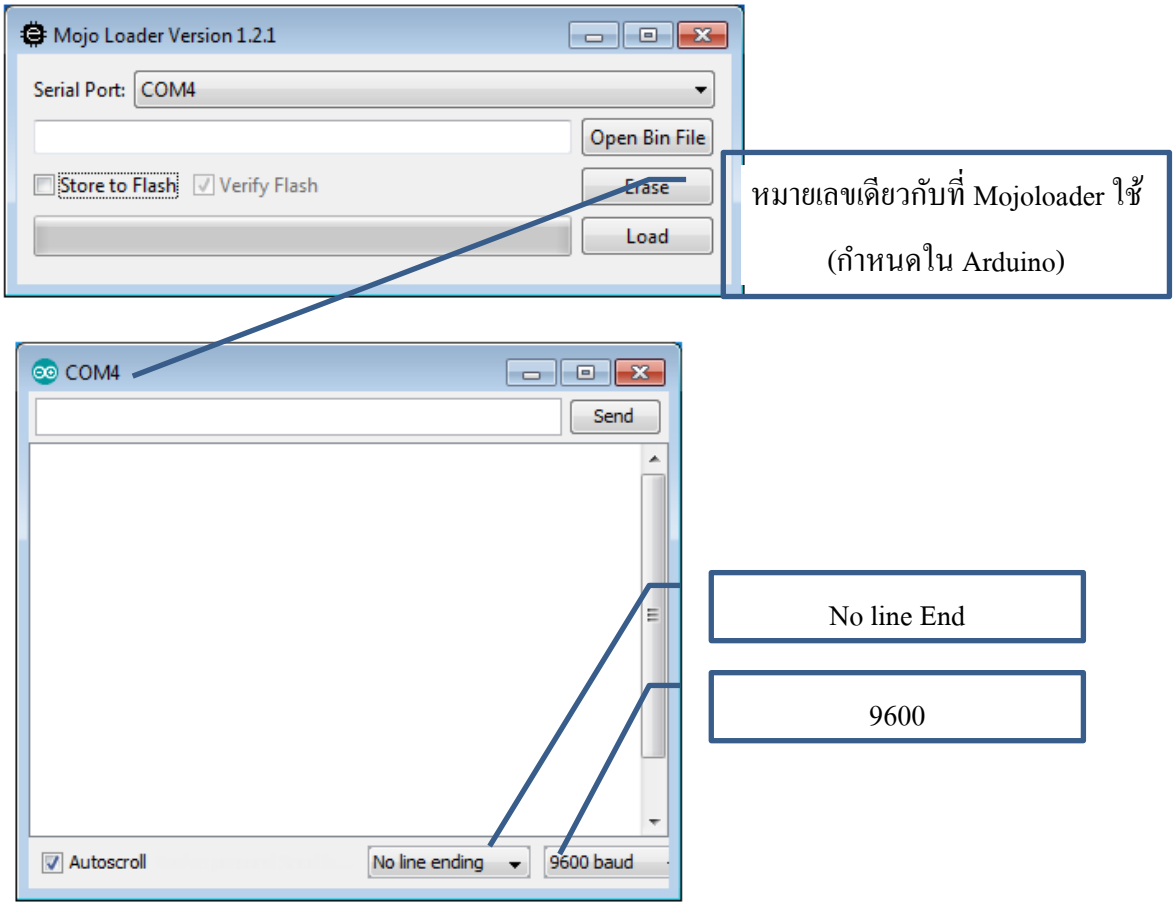
การทดลองรับข้อมูลจากพอร์ทอนุกรม

วัตถุประสงค์

รับข้อมูลจากพอร์ทอนุกรมแล้วแสดงเป็นรหัส ASCII ออกทาง LED ดวง

การทดลอง

1. ให้แก้ไฟล์ Mojo_top.vhd ดังนี้
2. ทำการแปลโปรแกรมจนได้ไฟล์ bin แล้วนำไปโปรแรมลงชิพ FPGA บนบอร์ด MOJO
3. ปิดโปรแกรม Mojo Loader เปิดโปรแกรม terminal (ในตัวอย่างนี้ใช้โปรแกรม Serial Monitor ของโปรแกรม Arduino 1.6.2) เพื่อใช้ทดลองการทำงาน สำหรับ Serial Port จะใช้ COM อะไร ให้ดูจาก Mojo Loader ให้ใช้พอร์ทเดียวกัน ส่วน Baudrate ใช้ 9600
4. เมื่อพิมพ์ตัวอักษรอะไรลงในช่องเพื่อส่งข้อมูลแล้วกด Enter ข้อมูลจะถูกส่งออกจาก PC ผ่านพอร์อนุกรม (USB) เข้าสู่บอร์ด Mojo ผ่านชิพ AVR เข้าสู่ FPGA ค่า ASCII ของอักขระที่พิมพ์ จะถูกนำแสดงทาง LED 8 ดวง บนบอร์ด Mojo เช่นถ้าพิมพ์เลข 1 แล้วส่ง จะได้รหัส 31H หรือ 0011 0001b



ไฟล์ Mojo_top.vhd ที่แก้ไข

```
----------------------------------------------------------------------------------
-- Mojo_top VHDL
-- Translated from Mojo-base Verilog project @ http://embeddedmicro.com/frontend/files/userfiles/files/Mojo-Base.zip
-- by Xark
--
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity serial2led is
    port (
        clk         : in  std_logic;          -- 50Mhz clock
        rst_n       : in  std_logic;          -- "reset" button input (negative logic)
        cclk        : in  std_logic;          -- configuration clock (?) from AVR (to detect when AVR ready)
        led         : out std_logic_vector(7 downto 0); -- 8 LEDs on Mojo board
        spi_sck     : in  std_logic;          -- SPI clock to from AVR
        spi_ss      : in  std_logic;          -- SPI slave select from AVR
        spi_mosi    : in  std_logic;          -- SPI serial data master out, slave in (AVR -> FPGA)
        spi_miso    : out std_logic;          -- SPI serial data master in, slave out (AVR <- FPGA)
        spi_channel : out std_logic_vector(3 downto 0);  -- analog read channel (input to AVR service task)
        avr_tx      : in  std_logic;          -- serial data transmited from AVR/USB (FPGA recieve)
        avr_rx      : out std_logic;          -- serial data for AVR/USB to receive (FPGA transmit)
        avr_rx_busy : in  std_logic           -- AVR/USB buffer full (don't send data when true)
    );
end serial2led;

architecture RTL of serial2led is

signal rst  : std_logic;          -- reset signal (rst_n inverted for postive logic)

-- signals for avr_interface
signal channel         : std_logic_vector(3 downto 0);
signal sample          : std_logic_vector(9 downto 0);
signal sample_channel: std_logic_vector(3 downto 0);
signal new_sample        : std_logic;
signal tx_data         : std_logic_vector(7 downto 0);
signal rx_data         : std_logic_vector(7 downto 0);
signal new_tx_data   : std_logic;
signal new_rx_data   : std_logic;
signal tx_busy         : std_logic;

-- signals for UART echo test
signal uart_data       : std_logic_vector(7 downto 0); -- data buffer for UART (holds last recieved/sent byte)
signal data_to_send  : std_logic;          -- indicates data to send in uart_data

-- signals for sample test
signal last_sample     : std_logic_vector(9 downto 0);

begin

rst <= NOT rst_n;                 -- generate non-inverted reset signal from rst_n button

-- NOTE: If you are not using the avr_interface component, then you should uncomment the
--       following lines to keep the AVR output lines in a high-impeadence state.  When
--       using the avr_interface, this will be done automatically and these lines should
--       be commented out (or else "multiple signals connected to output" errors).
--spi_miso <= 'Z';                    -- keep AVR output lines high-Z
--avr_rx <= 'Z';                      -- keep AVR output lines high-Z
--spi_channel <= "ZZZZ";              -- keep AVR output lines high-Z
```

```
-- instantiate the avr_interface (to handle USB UART and analog sampling, etc.)
avr_interface : entity work.avr_interface
    port map (
        clk         => clk,             -- 50Mhz clock
        rst         => rst,             -- reset signal
        -- AVR MCU pin connections (that will be managed)
        cclk        => cclk,
        spi_miso    => spi_miso,
        spi_mosi    => spi_mosi,
        spi_sck     => spi_sck,
        spi_ss      => spi_ss,
        spi_channel => spi_channel,
        tx          => avr_rx,
        tx_block    => avr_rx_busy,
        rx          => avr_tx,
        -- analog sample interface
        channel     => channel,         -- set this to channel to sample (0, 1, 4, 5, 6, 7, 8, or 9)
        new_sample  => new_sample, -- indicates when new sample available
        sample_channel => sample_channel,   -- channel number of sample (only when new_sample = '1')
        sample      => sample,          -- 10 bit sample value (only when new_sample = '1')
        -- USB UART tx interface
        new_tx_data  => new_tx_data,-- set to set data in tx_data (only when tx_busy = '0')
        tx_data      => tx_data,        -- data to send
        tx_busy      => tx_busy,        -- indicates AVR is not ready to send data
        -- USB UART rx interface
        new_rx_data  => new_rx_data,-- set when new data is received
        rx_data      => rx_data         -- received data (only when new_tx_data = '1')
    );
-- The following is simple test that will transmit any bytes received on UART and show upper 8-bits of analog 0
sample on LEDs
echo: process(clk, rst)
begin
    if rst = '1' then                           -- set signals to default on reset
--      tx_data       <= (others => '0');
        uart_data     <= (others => '0');
--      data_to_send <= '0';
--      new_tx_data  <= '0';
--      channel       <= "0000";                -- use channel 0 for this test
--      last_sample <= (others => '0');
--      led <= "11100111";                      -- flash LEDs on reset
    elsif rising_edge(clk) then
--      if data_to_send = '1' and tx_busy = '0' then   -- if there is data to send and UART is not busy then
--          tx_data <= uart_data;                   -- set tx_data input
--          new_tx_data <= '1';                     -- signal there is data to tx
--          data_to_send <= '0';                    -- clear our send flag
--      else
--          new_tx_data <= '0';                     -- else, we have no new tx data
--      end if;

        if new_rx_data = '1' then                   -- if there is new received data then
            uart_data <= rx_data;                   -- put it into uart_data register
--          uart_data <= "00111001";                -- put it into uart_data register
--          uart_data <= to_integer(last_sample(9 downto 2));
--          data_to_send <= '1';                    -- set flag that we have data to send
        end if;
--      if new_sample = '1' then                    -- if there is a new sample available then
--          last_sample <= sample;                  -- put it into last_sample register
--      end if;
--      led <= last_sample(9 downto 2);             -- display upper 8-bits of last 10-bit analog sample reading
        led <= uart_data;
    end if;
end process echo;

end RTL;
```