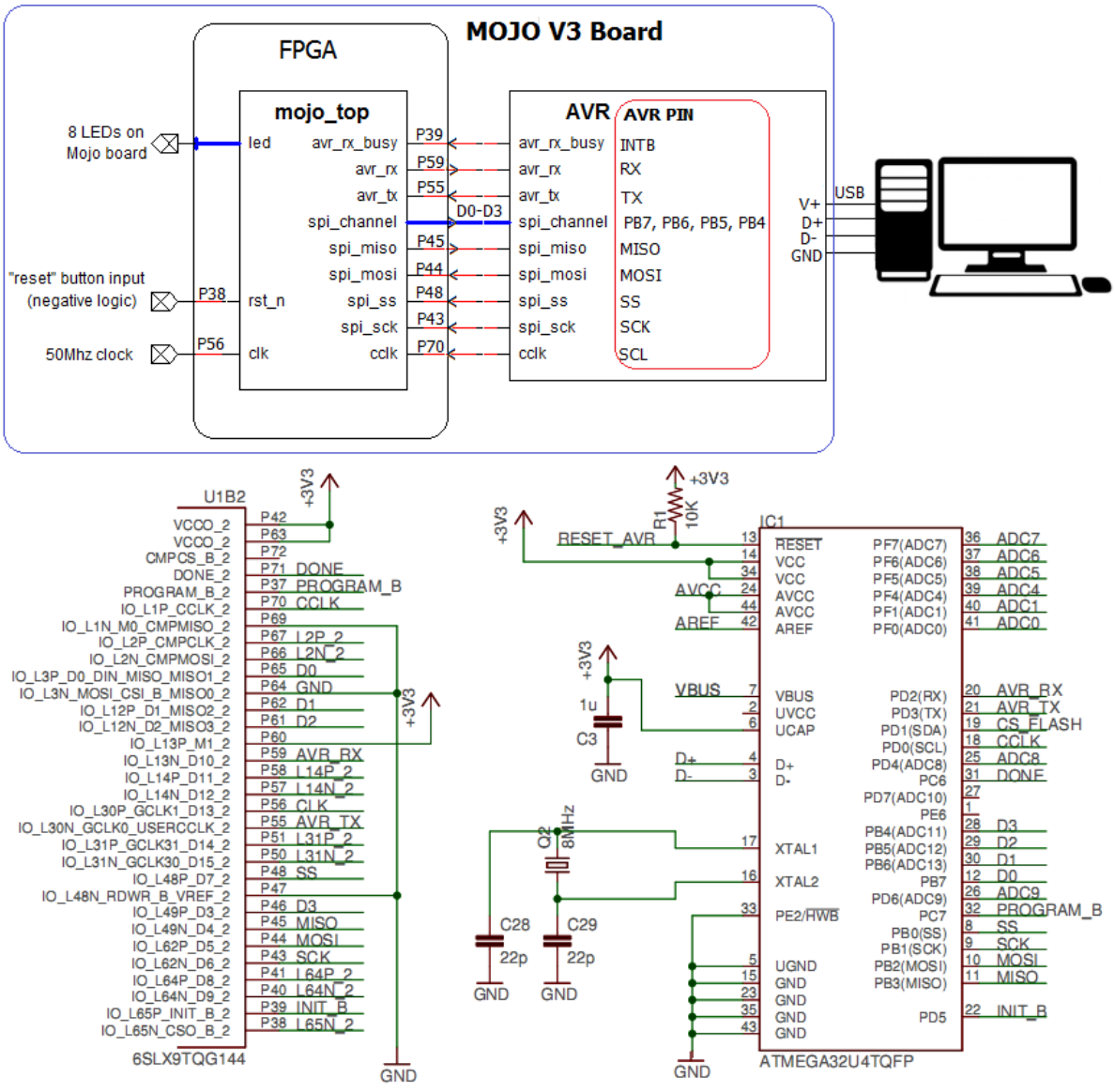
AVR interface

บล็อกไดอะแกรม

บอร์ด MOJO ใช้ชิป ATmega32u4 (3.3V, 8MHz) โดยทำหน้าที่หลักเป็นตัวโปรแกรมชิป FPGA และสามารถเชื่อมต่อผ่านขา Tx/Rx และ SPI กับ FPGA ได้ โมดูลสำหรับการเชื่อมต่อระหว่าง FPGA กับ ATmega32u4 คือ avr_interface ซึ่งอยู่ในตัวอย่างโปรแกรม Mojo-Base-VHDL-1.1.zip จาก http://sourceforge.net/projects/mojovhdl/

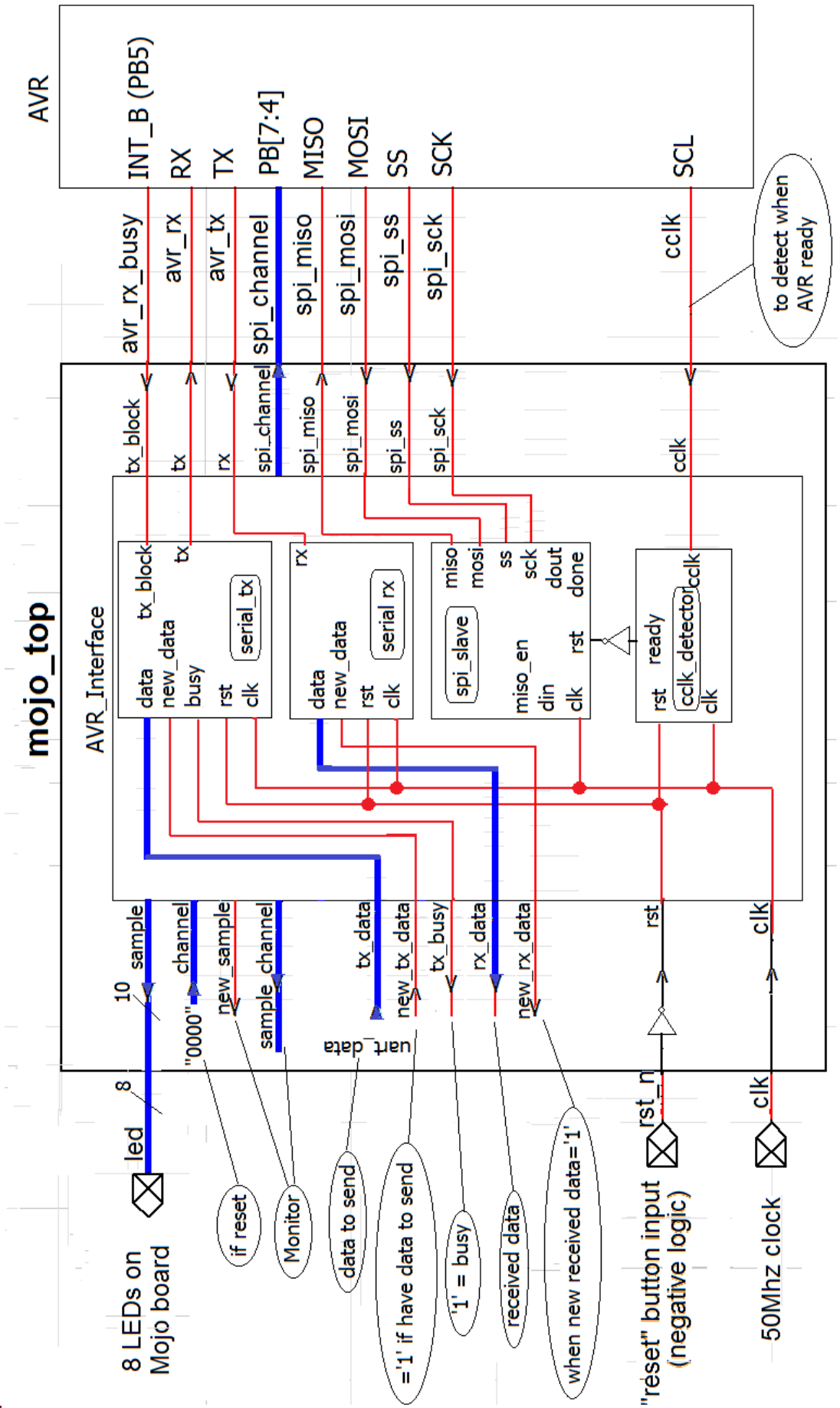
avr_interface มีรายละเอียดดังบล็อกไดอะแกรมตามรูป



avr_interface ประกอบด้วยโมดูลย่อยๆ

(1) spi_slave (2) serial_tx

(3) serial_rx (4) cclk_detector


AVR
INT_B (PB5)
RX
TX
PB[7:4]
MISO
MOSI
SS
SCK
SCL
avr_rx_busy
avr_rx
avr_tx
spi_channel
spi_miso
spi_mosi
spi_ss
spi_sck
cclk
to detect when AVR ready
mojo_top
AVR_Interface
tx_block
tx
rx
data
new_data
busy
rst
clk
serial_tx
serial rx
miso
mosi
ss
sck
dout
done
spi_slave
miso_en
din
ready
cclk_detector
sample
channel
new_sample
sample_channel
tx_data
new_tx_data
tx_busy
rx_data
new_rx_data
uart_data
"0000"
10
8
led
8 LEDs on Mojo board
if reset
Monitor
data to send
='1' if have data to send
'1' = busy
received data
when new received data='1'
rst_n
"reset" button input (negative logic)
50Mhz clock

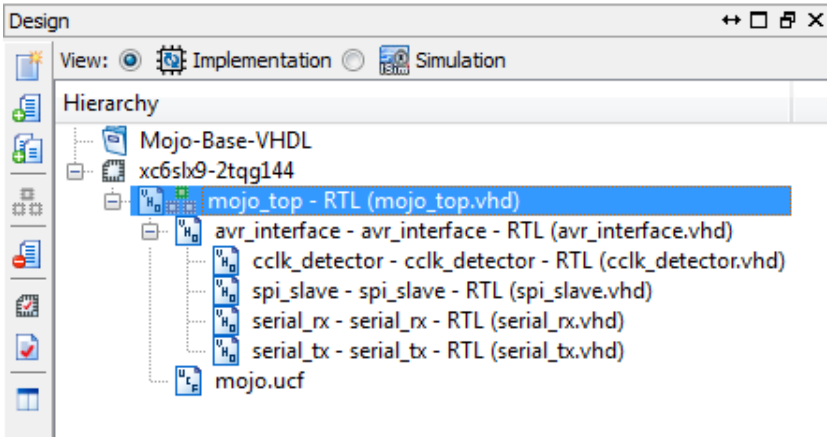
ทั้งนี้ให้ลองสังเกต การเรียกใช้โมดูลย่อยเหล่านี้ของภาษา VHDL รุ่นนี้ไม่ได้ใช้คำสั่ง component แต่ใช้การเรียกใช้จาก library work และบอกการเชื่อมต่อสัญญาณพร้อมเลย

```
avr_interface : entity work.avr_interface
 port map (.....
        .....
        .....
        );
```

ดังนั้นเราสามารถรับ-ส่งข้อมูลระหว่าง FPGA กับ PC ผ่านทางพอร์ทอนุกรมได้

สำหรับ ตัวอย่างการใช้งานให้ดูจากตัวอย่าง Mojo-Base โดยทำดังนี้

1. ให้ไป ดาวน์โหลดโปรแกรม Mojo-Base-VHDL-1.1.zip จาก http://sourceforge.net/projects/mojovhdl/
2. ขยายไฟล์ จะได้โฟลด์เดอร์ Mojo-Base-VHDL
3. ใช้ ISE เปิด Project Mojo-Base-VHDL.xise ซึ่งอยู่ใน sub flodder Mojo-Base-VHDL\ise_files หรือดับเบิลคลิกไฟล์ Mojo-Base-VHDL.xise ใน sub flodder Mojo-Base-VHDL\ise_files เพื่อรัน ise ในหน้าต่าง Design จะปรากฏไฟล์ต่างๆดังนี้



ไฟล์ mojo_top.vhd เป็นตัวอย่างการใช้งาน avr_interface-avr_interface.vhd

4. ให้ทดลองแปลโปรแกรมจนได้ mojo_top.bin แล้วนำไปโปรแกรมลงชิพ FPGA บนบอร์ด MOJO ด้วย
5. ให้ใช้ VR ต่อวงจรดังนี้เข้ากับบอร์ด MOJO

6. ทดลองปรับค่า VR แล้วสังเกต การติดของ LED 8 ดวง ซึ่งเป็นค่า 8 บิตบน ของค่า analog ที่ป้อนเข้าขา A0 ของ AVR

ในตัวอย่างต่อๆไป จะใช้ไฟล์ mojo_top.vhd นี้มาดัดแปลงเขียนเป็นโมดูลแบบต่างๆ

```
--------------------------------------------------------------------------------
-- 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 mojo_top 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 mojo_top;
```

กำหนดขาสัญญาณ

```
architecture RTL of mojo_top 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);
```

1. signals for avr_interface
2. signals for UART echo test
3. signals for sample test (A/D)

```
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')
    );
```

เรียกใช้โมดูล avr_interface และทำการเชื่อมต่อสัญญาณเข้ากับระบบ

```
-- 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 <= "11111111"; -- 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
            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
    end if;
end process echo;
```

เฉพาะในส่วนนี้

โปรแกรมที่เป็นสีแดงคือส่วนที่ใช้ติดต่อกับพอร์ทอนุกรม

โปรแกรมสีน้ำเงิน ใช้ติดต่อกับพอร์ทอนาลอก

```
end RTL;
```