Design of the Washu-2 Processor – Part 3

- Supporting Components

Jon Turner
The WashU-2 on the S3 board

- VGA display connector
- Push buttons:
  - reset
  - single step
  - load data into memory
- Display showing:
  - internal registers
  - memory locations
- Knob for entering data
- Slide switches
Using the WashU-2 Processor

- Reset by setting pressing “north” button
  - restarts processor, but does not re-initialize memory
  - re-program device to re-initialize memory

- Using single step mode
  - press west button to enter single-step mode, press again to execute one instruction
  - press south button to exit single step mode

- Using display
  - display shows
    IREG  ACC  SnoopAdr
    PC    IAR  SnoopData
  - when switch(3)=0, knob controls Snoop Address register
  - when switch(3)=1, knob controls Snoop Data register
  - press down on east button to write snoopData value
Support Components for Proto Board

- **dispBuf**
  - hSync, vSync, dispVal,

- **Processor**
  - PC, ACC, REG, IAR
  - EN/RW, EN/RW

- **Console**
  - regSel, cpuReg, pause, reset
  - snoopAddr, snoopData

- **Lcd Signals**
  - knobIntf, debouncer

- **Icd Display**

**Display buffer drives VGA display interface**

**Console includes several internal processes**
- displays CPU registers and snoop registers
- implements single-step operation by pausing processor
- implements snooping and memory update by accessing memory (pausing processor as it does so)
Knobs and Snooping

knob changes increment snoopAdr, then snoopData

button press triggers write to M[10], program then updates sum

input word and sum
Simulation – Single Step Operation

- Button press to do one instruction
- Execute one, then pause again
- Pause drops to allow one instruction
- Button press to resume
- Back to normal
VHDL for Console

define entity console is port(
    clk: in std_logic;
    btn: in buttons;
    knob: in knobSigs;
    swt: in switches;
    led: out LEDs;
    resetOut: out std_logic;
    pause: out std_logic; -- pause CPU
-- memory signals
    memEnIn, memRwIn: in std_logic;
    memEnOut, memRwOut: out std_logic;
    aBus: out word;
    dBus: inout word;
-- signals for observing CPU registers
    regSelect: out std_logic_vector(1 downto 0);
    cpuReg: in word;
-- signals for controlling LCD display
    lcd: out lcdSigs);
end console;
architecture a1 of console is

-- signals for controlling input from knob, buttons
signal dBtn, prevDBtn: buttons;
signal reset: std_logic;
signal tick, clockwise: std_logic;
signal delta: word;
-- single step control signal
signal singleStep: std_logic;
-- local signals for controlling memory
signal memEn, memRw: std_logic;
-- signals for controlling snooping
signal snoopAddr: address; signal snoopData: word;
signal snoopMode, snoopTime, writeReq: std_logic;
signal snoopCnt: std_logic_vector(6*operationMode + 9 downto 0);

-- signals for controlling lcd display
constant CNTR_LENGTH: integer := 8 + operationMode*12;
signal lcdCounter: std_logic_vector(CNTR_LENGTH-1 downto 0);
signal lowBits: std_logic_vector(CNTR_LENGTH-6 downto 0);
signal update: std_logic;
signal selekt: std_logic_vector(4 downto 0);
signal nuchar: std_logic_vector(7 downto 0);
type hex2asciiMap is array(0 to 15) of character;
constant hex2ascii: hex2asciiMap :=
( 0 => '0', 1 => '1', 2 => '2', 3 => '3', 4 => '4',
  5 => '5', 6 => '6', 7 => '7', 8 => '8', 9 => '9',
  10 => 'a', 11 => 'b', 12 => 'c', 13 => 'd', 14 => 'e', 15 => 'f');
begin
  snoopMode <= swt(3);  -- when low, knob controls snoop address,
                    -- when high, snoop data
  reset <= dBtn(0);    resetOut <= reset;
  -- connect all the sub-components
  db: debouncer generic map(width => 4) port map(clk, btn, dBtn);
  kint: knobIntf port map(clk, reset, knob, tick, clockwise, delta);
  disp: lcdDisplay port map(clk, reset, update, selekt, uchar, lcd);

  pause <= singleStep or snoopTime;
  -- process for controlling single step operation
  process(clk) begin
    if rising_edge(clk) then
      prevDBtn <= dBtn;
      if reset = '1' then singleStep <= '0';
    else
      if dBtn(3) > prevDBtn(3) then singleStep <= not singleStep;
      elsif dBtn(3) = '1' then singleStep <= '1';
      elsif dBtn(2) > prevDBtn(2) then singleStep <= '0';
      end if;
    end if;
  end process;

  memEnOut <= memEnIn or memEn;
  memRwOut <= memRwIn and memRw;
end

pause CPU when single-stepping or when snooping

enter single-step mode or advance one instruction when btn(3) is pressed

exit single-step mode when btn(2) is pressed

propagate memory signals from processor when not snooping
process(clk) begin
  if rising_edge(clk) then
    if reset = '1' then
      snoopAdr <= (others => '0'); snoopData <= '0';
      snoopCnt <= (others => '0'); writeReq <= '0';
    else
      snoopCnt <= snoopCnt + 1;
      if dBtn(1) > prevDBtn(1) and snoopMode = '1' then
        writeReq <= '1';
      end if;
      if writeReq = '1' and snoopTime = '1'
        and snoopCnt(3 downto 0) = x"f" then
        writeReq <= '0';
      end if;
      if snoopTime = '1' and snoopMode = '0' then
        if snoopCnt(3 downto 0) = x"d" then snoopData <= dBus;
      end if; end if;
    if tick = '1' then
      if snoopMode = '0' then
        if clockwise = '1' then snoopAdr <= snoopAdr + delta;
        else snoopAdr <= snoopAdr - delta;
      end if;
      else
        if clockwise = '1' then snoopData <= snoopData + delta;
        else snoopData <= snoopData - delta;
      end if; end if; end if; end if; end process;
snoopTime <= '1' when snoopCnt(high downto 4) = (snoopCnt(high downto 4) => '1')
else '0';

-- process controlling memory signals for snooping
process (snoopTime, snoopCnt, snoopData, snoopAdr) begin
    memEn <= '0'; memRw <= '1';
    aBus <= (others => 'Z'); dBus <= (others => 'Z');
    if snoopTime = '1' then
        -- allow time for in-progress instruction to complete
        if snoopCnt(3 downto 0) = x"c" then
            memEn <= '1'; aBus <= snoopAdr;
        elsif writeReq = '1' and snoopCnt(3 downto 0) = x"f" then
            memEn <= '1'; memRw <= '0';
            aBus <= snoopAdr; dBus <= snoopData;
        end if;
    end if;
end process;

perform memory read and write
regSelect <= "00" when selekt <= slv(4,5) else 
    "10" when selekt <= slv(10,5) else 
    "01" when selekt <= slv(20,5) else 
    "11";

lowBits <= lcdCounter(CNTR_LENGTH-6 downto 0);
update <= '1' when lowBits = (lowBits'range => '0') else '0';
selekt <= lcdCounter(CNTR_LENGTH-1 downto CNTR_LENGTH-5);

process (cpuReg, snoopAdr, snoopData, selekt) begin
    case selekt is
        -- high nibble of processor registers
        when "00000" | "00110" | "10000" | "10110" => 
            nuChar <= c2b(hex2Ascii(int(cpuReg(15 downto 12))));
        -- second nibble of processor registers
        when "00001" | "00111" | "10001" | "10111" => 
            nuChar <= c2b(hex2Ascii(int(cpuReg(11 downto 8))));
        ...

        -- nibbles of snoopAdr register
        when "01100" => nuChar< c2b(hex2Ascii(int(snoopAdr(15 downto 12))));
        when "01101" => nuChar< c2b(hex2Ascii(int(snoopAdr(11 downto 8))));
        ...

        -- nibbles of snoopData register
        when "11100" => nuChar< c2b(hex2Ascii(int(snoopData(15 downto 12))));
        when "11101" => nuChar< c2b(hex2Ascii(int(snoopData(11 downto 8))));
        ...
    end case;
end process;
Exercises

1. Suppose we put the processor in single-step mode while the processor is executing the machine instruction 0103, at memory location 13c4. What values will appear on the LCD display in the first four positions of both lines?

2. Draw a timing diagram that shows what happens during one snoopTime interval when snoopMode is low. Your diagram should show the memory signals, the snoop registers, the snoopCnt signal and the pause signal. Assume that initially snoopAdr=1234, snoopData=abcd and that M[1234]=3210.

3. Draw a timing diagram that shows what happens during one snoopTime interval when snoopMode is high. Your diagram should show the memory signals, the snoop registers, the snoopCnt signal and the pause signal. Assume that initially snoopAdr=1234, snoopData=abcd and that M[1234]=3210.

4. When the console’s memEn signal is high, what can we say about the value of snoopCnt? What is the processor’s state at this moment? When the console’s memRw signal is high, what can we say about the value of snoopCnt? Explain why memEnIn and memEn are never high at the same time. Explain why memRwIn and memRW are never low at the same time.

5. The console requires that you push button 1, whenever you want to store the current snoopData value in memory. An alternative version would track the value in snoopData continuously. So, when switch 3 is high, the memory would be updated whenever snoopData changes. Explain how you would implement this.