University of Glasgow College UESTC

**Digital Circuit Design 3**

Laboratory Session

## 1. Aims & Objectives

The aim of this laboratory is to gain experience in solving a realistic digital design problem. Specifically we consider the design of a crossbar switch system to route arbitrary length 8-bit word data packets – an actual design problem encountered in this Department during research into construction of parallel computer systems some years ago. Our objectives are to :

- Gain an understanding of crossbar switch architectures and packet data routing.
- Develop the skills of applying a CAD tool (i.e. Vivado) workflow to realistic digital design.
- Deepen skills in writing clear, synthesizable VHDL.
- Practice creation of test vectors for realistic designs.

## 2. Crossbar Switch Design & Data Packets

Communication between modules in a computing system usually makes use of a common bus. However, when a number of different modules might need to communicate simultaneously (for instance one CPU talking to a graphics card, whilst an I/O controller talks to main memory) then common busses are inefficient; only one piece of data can be on the bus at any time, and a number of clock cycles are often wasted negotiating which modules may access the bus. For such systems, the additional logic and wiring of a crossbar switch may be worthwhile.

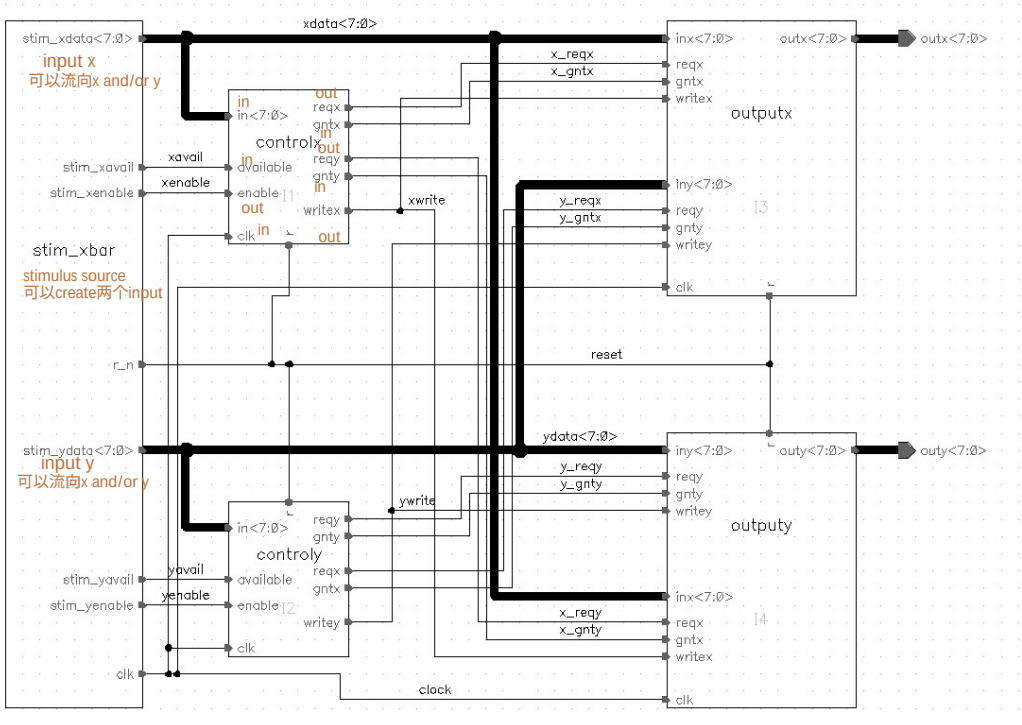


Figure 1 – Schematic of Crossbar Switch

Figure 1 above shows such a switch with two input data paths (`stim_xdata`, and `stim_ydata` which are both here created by the stimulus source `stim_xbar`) and two output data paths (`outx`, `outy`). Data from the x-input can flow to the x and/or y outputs. Data from the y-input can flow to the x and/or y outputs. It is possible for x⇒x and y⇒y transfers to be carried out simultaneously. It is also possible for x⇒y and y⇒x transfers to occur simultaneously, or for one input to be broadcast to both outputs. This flexibility, controlled by logic in the `control` and `output` blocks, gives crossbar switches their usefulness.

You will be given the VHDL for both the stimulus block `stim_xbar`, the output logic in `outputx`/`outputy`, and a template of the logic in `controlx`/`controly`. You will be required to complete and test the VHDL for `controlx`/`controly`.

The crossbar switch hardware must be given appropriate information on how data on any input should be routed. It would be wasteful, and for any design of complexity, impossible, for each transmitted word (in figure 1 each word is 8 bits wide) to contain both routing information and data. Therefore bytes are usually grouped into large packets which can be transmitted to the same destination a byte at a time, with a few bytes at the beginning of each packet (the packet *header*) giving the routing information for the whole packet.
Likewise, it is important to know when one packet ends and another begins. The length of each packet may be fixed, or given in the packet header, or a special *end-byte* may be used to signify the end of a packet (and therefore that the next byte will be the first byte of the header of the next packet).

In this example we use perhaps the simplest packet encapsulation. A single header byte is used, and only the lowest order bit of this byte need be considered. If the lowest order bit of the header byte is a '0', then the packet should be routed to `outx`. Otherwise the packet should be routed to `outy`.
The end of a packet is likewise signified by a single byte - $FF_h$. If the byte $FF_h$ is seen, the next byte is the header byte for the next packet.

# 3. Developing the Design

## 3.1. Copying and Reviewing the Lab Files

Two files have to be copied over for this laboratory session. Make a new directory `esdlab2` under your directory. Change to this directory and copy files `top_tb.vhd` and `control_template.vhd` into this directory. `top_tb.vhd` describes the operation of stimulus block and the logic of the state machine. Draw the state machine described by `top_tb.vhd` in your lab record and describe its operation.

## 3.2. Design & Test

The core task of this laboratory in lab sessions 3&4 is to use the insight gained from an analysis of figure 1 and the pre-designed modules to:

1. Construct a state machine for `controlx` / `controly`.
2. Translate that state machine into correct VHDL.
3. Test the completed module using the simulation tools of the CAD design suite, and the preliminary set of test vectors described in `top_tb.vhd`.
4. Develop a comprehensive set of test vectors to test the operation of the crossbar switch.

As a design exercise you will be expected to make most of the design and coding decisions in your lab group, and confirm your decisions using the CAD development and simulation tools. Feel free, however, to call upon the advice and help of laboratory demonstrators whenever you need advice or guidance. Your completed VHDL code for `controlx` / `controly`, should give a crossbar switch that is functionally correct, easy to synthesize, and retains maintainability. Improved test vectors for `top_td.vhd` should cover all the possible realistic inputs that might be applied to the crossbar switch.

### 3.3. Report

The design exercise should then be written-up as an individual laboratory report, which will be worth 15% of the final degree examination results.

**Attachment I: top_tb.vhd**

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date: 2018/10/26 15:40:42
-- Design Name:
-- Module Name: top_tb - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all; -- for internal counter etc.


-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;
```

```
-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity top_tb is
end top_tb;

architecture behavioral of top_tb is
        component control
                PORT(
        enable : OUT std_logic;
        reqx : OUT std_logic;
        reqy : OUT std_logic;
        write : OUT std_logic;
        available : IN std_logic;
        clk : IN std_logic;
        data_in : IN std_logic_vector(7 DOWNTO 0);
        gntx : IN std_logic;
        gnty : IN std_logic;
        reset : IN std_logic
      );

        end component;
        --INPUT
        signal clk: std_logic;
        signal r_n: std_logic;
        signal stim_xdata: std_logic_vector(7 downto 0);
        signal stim_ydata: std_logic_vector(7 downto 0);
```

```
        signal stim_xavail: std_logic :='0';
        signal stim_yavail: std_logic :='0';
        signal gntx: std_logic :='0';
        signal gnty: std_logic :='0';
        --OUTPUT
        signal stim_xenable: std_logic :='0';
        signal stim_yenable: std_logic :='0';
        signal writex: std_logic :='0';
        signal writey: std_logic :='0';
        signal reqx: std_logic :='0';
        signal reqy: std_logic :='0';

        signal data_out: std_logic_vector(7 downto 0);
        --VARIABLE
        ----stim_xbar.vhd
        signal internalclock : std_logic; -- internal clock
        signal dummy : std_logic; -- dummy signal
        ----output.vhd
        type states is (poll_x, poll_y, grant_x, grant_y);
        -- Present state.
        signal present_state : states;


begin

        controlx:control port map(
    --Input
                clk=>clk,
                reset=>r_n,
```

```
            data_in=>stim_xdata,
            available=>stim_xavail,
            gntx=>gntx,
            gnty=>gnty,
            --Output
            enable=>stim_xenable,
            write=>writex,
            reqx=>reqx,
            reqy=>reqy
);
   controly:control port map(
--Input
            clk=>clk,
            reset=>r_n,
            data_in=>stim_ydata,
            available=>stim_yavail,
            gntx=>gntx,
            gnty=>gnty,
            --Output
            enable=>stim_yenable,
            write=>writey,
            reqx=>reqx,
            reqy=>reqy
);

   resetgen : process
   begin -- process resetgen
            r_n <= '0';
            wait for 2450 ns;
```

```
        r_n <= '1';
-- Add more entries if required at this point !


        wait until dummy'event;
end process resetgen;


-- purpose: Generates the internal clock source. This is a 50% duty
-- cycle clock source, period 1000ns, with the first half of
-- the cycle being logic '0'.
--
-- outputs: internalclock
clockgen : process
begin -- process clockgen
        internalclock <= '0';
        wait for 500 ns;
        internalclock <= '1';
        wait for 500 ns;
end process clockgen;
-- purpose: Generates the stimuli for the data, using a counter based
-- approach which is much cleaner than explicit timings
--
-- outputs: stim_inst
-- stim_a
-- stim_b


datagen1 : process
-- Since we're going to output a number of different test
-- data we need an internal counter to keep track of things.
```

```
variable count : unsigned (5 downto 0) := "000000";
begin -- process datagen1
        wait until internalclock'event and internalclock = '1';
        count := count + 1;
        if count = 4 then -- simple data transfer x -> x
                stim_xavail <= '1'; -- data flag raised
                wait until internalclock'event and internalclock = '1'
                and stim_xenable = '1';

                stim_xdata <= "00000010"; -- header word pushed when system
                stim_xavail <= '0'; -- requests transfer, and flag back down

                wait until internalclock'event and internalclock = '1'
                and stim_xenable = '1';

                stim_xdata <= "10101010"; -- data word pushed

                wait until internalclock'event and internalclock = '1'
                and stim_xenable = '1';

                stim_xdata <= "11111111"; -- end word pushed
        elsif count = 5 then    -- NO OPERATION
                stim_xavail <= '0';
                stim_xdata <= "00000000";
        else  -- NO OPERATION
                stim_xavail <= '0';
                stim_xdata <= "00000000";
        end if;
end process datagen1;
```

```
-- ************************************

-- STEP 4: Develop a comprehensive set of test vectors to test the operation of the crossbar switch.

-- Add more test code here:




-- ************************************



-- now drive the output clock, this is simply the internal clock

-- nb: could also invert the clock if desired to allow for signals

-- to be generated pseudo-asynchronously


clk <= internalclock;


--output.vhd

process (clk, r_n)

begin

-- Activities triggered by asynchronous reset (active low).

        if (r_n = '0') then      reset=0

        -- Set the default state and outputs.

                present_state <= poll_x;        states这个type里有四个参数：poll_x, poll_y, grant_x, grant_y

                gntx <= '0';

                gnty <= '0';

                data_out <= "00000000";

        elsif (clk'event and clk = '1') then        clk等于1的时候状态不变
```

```
-- Set the default state and outputs.
        present_state <= poll_x;
        gntx <= '0';
        gnty <= '0';
        data_out <= "00000000";
        case present_state is
            when poll_x =>
                if (reqx = '1') then
                    present_state <= grant_x;
                else
                    present_state <= poll_y;
                end if;
            when poll_y =>
                if (reqy = '1') then
                    present_state <= grant_y;
                else
                    present_state <= poll_x;
                end if;
            when grant_x =>
                if (writex = '1') then
                    data_out <= stim_xdata;
                end if;
                if (reqx = '1') then
                    present_state <= grant_x;
                else
                    present_state <= poll_y;
                end if;
                gntx <= '1';
            when grant_y =>
```

大概环节就是present_state有不同的情况，case里解释了从poll_x开始的不同情况

poll-grant-req为1则dataout它的data

选择的语法

不同情况下的操作过程

轮询x

在poll_x的情况下

授予x

要求y则授予y

```
                    if (writey = '1') then
                        data_out <= stim_ydata;
                    end if;
                    if (reqy = '1') then
                        present_state <= grant_y;
                    else
                        present_state <= poll_x;
                    end if;
                    gnty <= '1';
                when others =>
                    -- Set the default state and outputs.
                    present_state <= poll_x;
                    gntx <= '0';
                    gnty <= '0';
                    data_out <= "00000000";
            end case;
        end if;
    end process;

end behavioral;
```

**Attachment II: control_template.vhd**

```
--------------------------------------------------------------------------------
-- Title : Routing control for a crossbar switch.
--------------------------------------------------------------------------------
-- Description: Controller for routing a packet to one of two outputs depending on the zeroth
--                  bit of the packet header word
--                  The whole packet is then sent out, stopping when the 'FF' word is sensed.
--                  The controller sends request signals to the appropriate output and sends the
--                   message on when it gets a grant signal in return.
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

architecture behavior of control is

-- Behaviour follows the 'classic' state machine method
   -- Possible states.

                              first in first out, 先入先出
   type states is (poll_fifo, raise_enable, check_data, setup_x, setup_y, data_xfer, data_yfer);

   -- Present state.

   signal present_state : states;

begin
```

```
-- Main process.

process (clk, reset)

begin

   -- Activities triggered by asynchronous reset (active low).

   if (reset = '0') then

      -- Set the default state and outputs.

      present_state <= poll_fifo;


   elsif (clk'event and clk = '1') then

      -- Set the default state and outputs.

      present_state <= poll_fifo;
      enable <= '0';
      reqx <= '0';
      reqy <= '0';
      write <= '0';

      case present_state is

         when poll_fifo =>
```

when raise_enable =>

when check_data =>

when setup_x =>

when setup_y =>

when data_xfer =>

when data_yfer =>

when others =>

```
            end case;

        end if;

    end process;

end behavior;
```