Glasgow College, UESTC

Digital Circuit Design

LAB for Crossbar Switch Design

Name: Xinyue Lan

UoG ID: 2357734L

UESTC ID: 2017200601018

University of Glasgow

电子科技大学 University of Electronic Science and Technology of China

1 Introduction

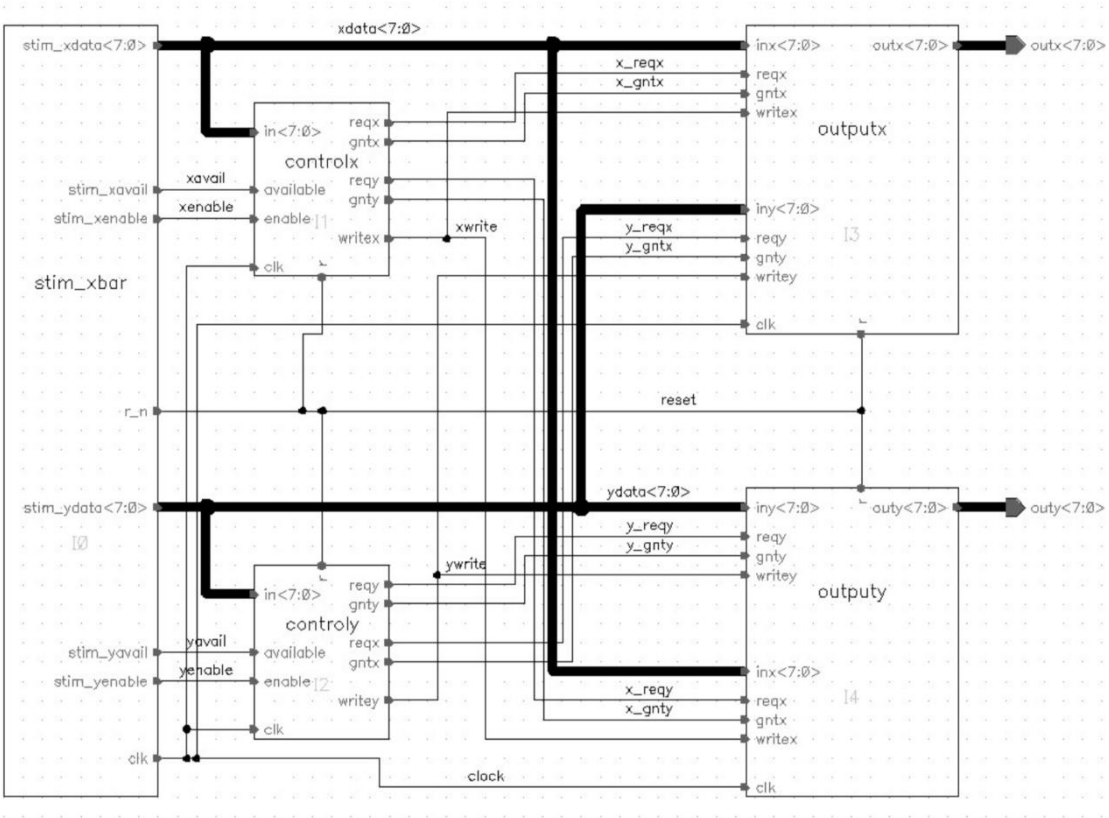
The purpose of this lab is to design a crossbar switch using VHDL to route data packets to output x/y via 2 control modules. Routing is essential in the field of communication, especially when data is transferred simultaneously from several transmitters to multiple receivers. In order to improve the efficiency of common bus in this situation, the additional logic and wiring of a crossbar switch may be worthwhile.

2 Method and design

2.1 Crossbar Switch

We explicitly applied (1) structural description by using component and its port map and (2) behavioral description by using several concurrent processes in the design of crossbar switch structure.

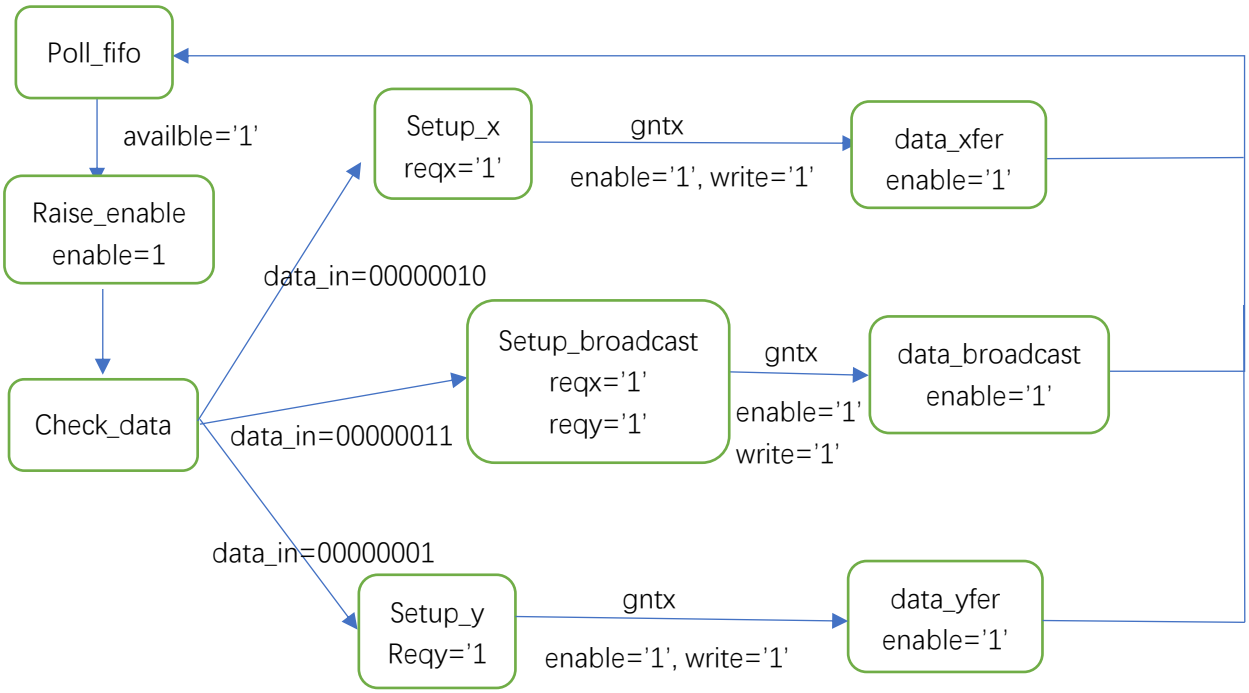
According to the figure describing schematic of the crossbar switch, the main purpose and principle of the crossbar switch can be figured out:



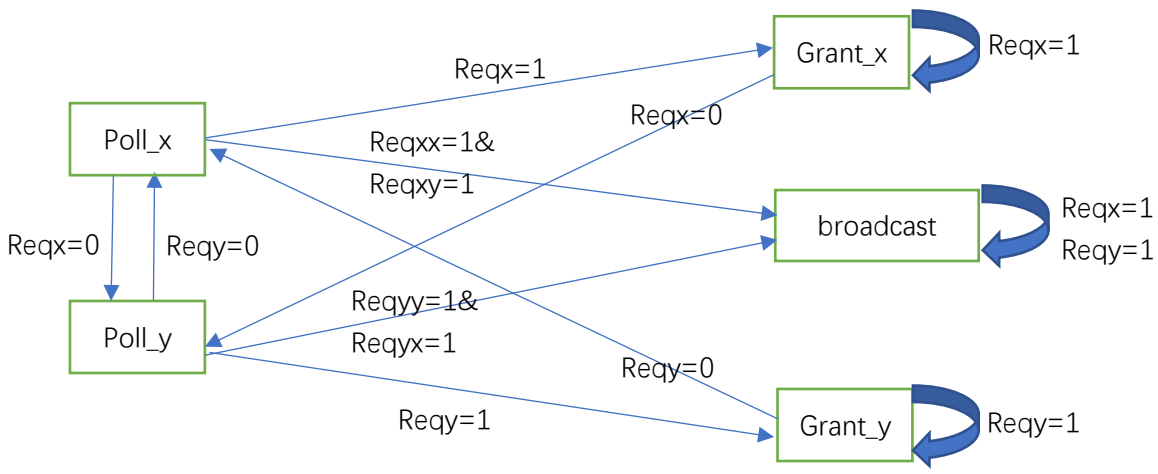
It is possible for x⇒x and y⇒y transfers to be carried out simultaneously. It is also possible for x⇒y and y⇒x transfers to occur simultaneously, or for one input to be broadcast to both outputs. This flexibility, controlled by logic in the

control and output blocks, gives crossbar switches their usefulness. In this lab, we use a simple way to encapsulate the packet: a single header byte is used to show the routing information and only the last two bits of this byte need be considered. If the last two bits of the header byte is '10', the packet should be routed to out x. If the last two bits of the header byte is '01', the packet will be routed to out y. If the last two bits of the header byte is '11', the packet will be broadcasted to both x and y. The end of a packet is likewise signified by another single byte. The final byte means the next received byte will be the header byte for the next packet

2.2 The State Machine Diagram of Control (control.vhd)



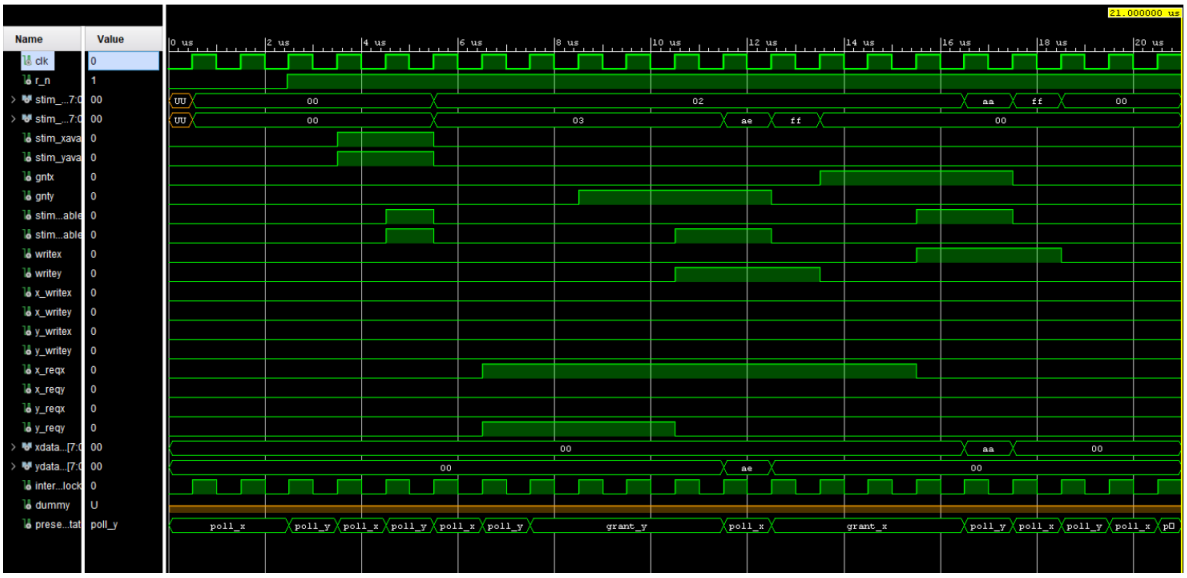
2.3 The State Machine Diagram of Receiver Side (top_tb.vhd)



3 System Test and Results

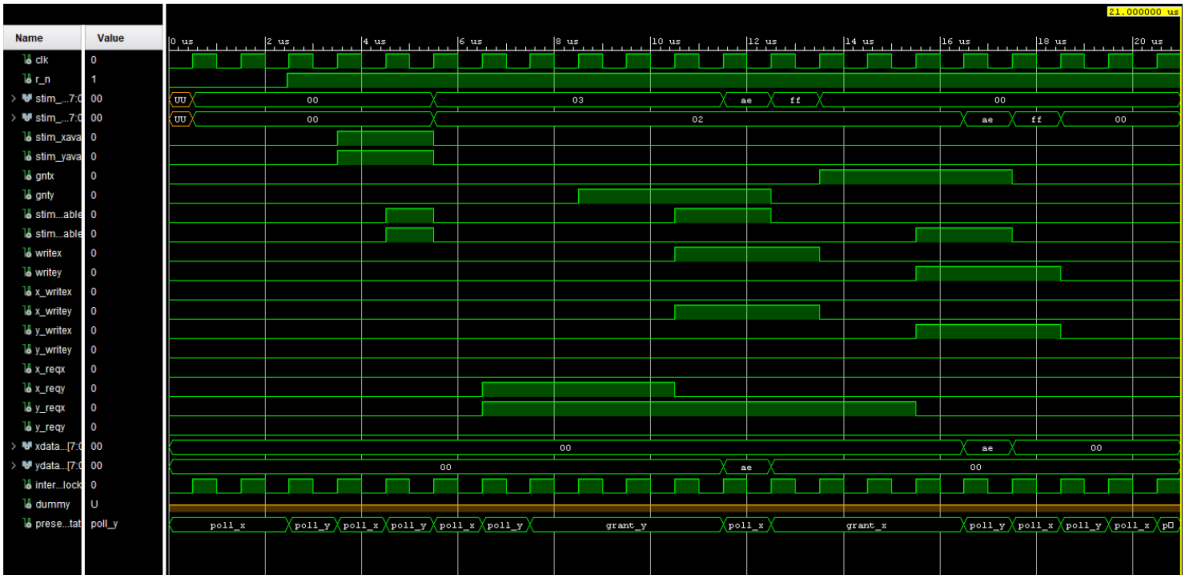
3.1 x to x and y to y

The headers of packages from input x and y are ‘00000010’ and ‘00000001’, respectively.



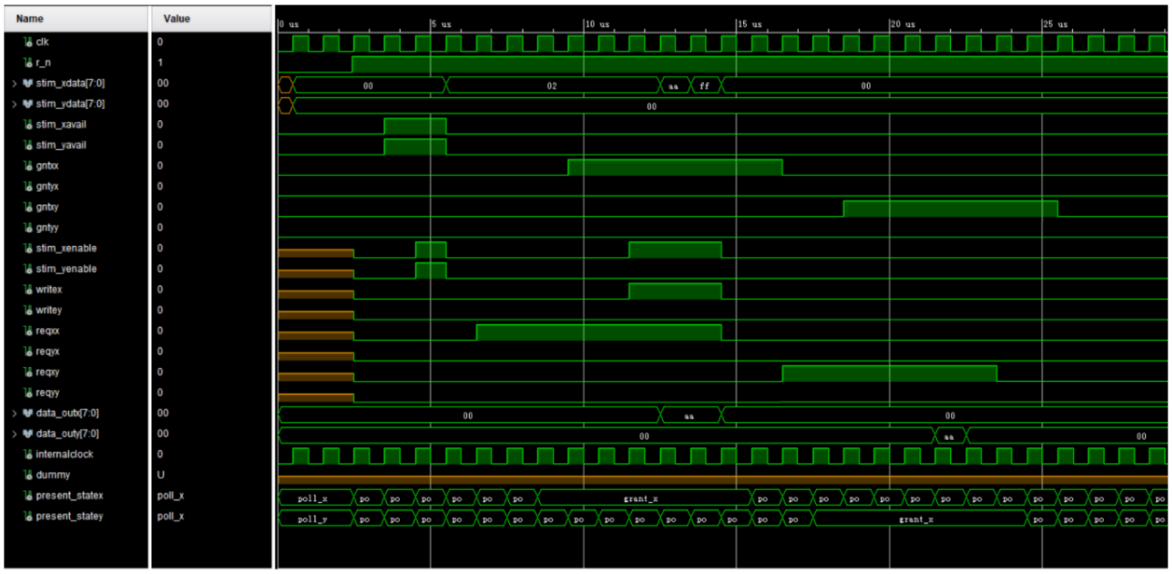
3.2 x to y and y to x

The headers of packages from input x and y are ‘00000001’ and ‘00000010’, respectively.



3.3 Broadcast

One input (i.e. stim_xdata) can also be transferred into control x unit and broadcast to both x and y outputs.



4 Conclusion

In this experiment, I designed a crossbar switch which can achieve the routing and transferring of data packets from one or two inputs to one or two outputs. I studied how to use diagram to present the change of state machine and how to program using VHDL language in order to simulate the crossbar switch I proposed. The result shows the feasibility of my design, which can handle each input and condition well. From this experiment, I am familiar with VHDL and understand how to use the state diagram to represent logic circuit and how to test my proposed system using VHDL coding.

APPENDIX

Control.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity control is
PORT(
enable : OUT std_logic;
reqx : OUT std_logic;
reqy : OUT std_logic;
writex : OUT std_logic;
writey : 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 control;
architecture behavior of control is
-- Behaviour follows the 'classic' state machine method
-- Possible states.
type states is (poll_fifo, raise_enable, check_data, setup_broadcast, data_broadcast, 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';
writex <= '0';
writey <= '0';
case present_state is

when poll_fifo =>
if available='1' then
present_state<=raise_enable;
end if;

when raise_enable =>
enable<='1';
present_state<=check_data;

when check_data =>
if (data_in(0)=’1’) then
```

```
present_state<=setup_y;
elsif (data_in(0)=’0’) then
present_state<=setup_x;
end if;

when setup_x=>
reqx<='1';
if gntx='1' then
enable<='1';
present_state<=data_xfer;
else
present_state<=setup_x;
end if;

when setup_y =>
reqy<='1';
if gnty='1' then
enable<='1';
present_state<=data_yfer;
else
present_state<=setup_y;
end if;

when data_xfer =>
enable<='1';
y_write<='1';
reqx<=’0’;
if (data_in = "11111111") then
```

```
enable<='0';
present_state<=poll_fifo;
else
present_state<=data_xfer;
end if;

when data_yfer =>
enable<='1';
x_write<='1';
reqy<=’0’
if (data_in = "11111111") then
enable<='0';
present_state<=poll_fifo;
else
present_state<=data_yfer;
end if;

when others =>
enable<='0';
reqx<='Z';
reqy<='Z';
x_write<='0';
y_write<='0';
end case;
end if;
end process;
end behavior;
```

end of control.vhd

top_tb.vhd

```
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;
x_write : OUT std_logic;
y_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 x_writex: std_logic :='0';
signal x_writey: std_logic :='0';
signal y_writex: std_logic :='0';
signal y_writey: std_logic :='0';
signal x_reqx: std_logic :='0';
signal x_reqy: std_logic :='0';
signal y_reqx: std_logic :='0';
signal y_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=>x_reqx,
reqy=>x_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=>y_reqx,
reqy=>y_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
-------------------------------------------------------------------------------------------------------------------------
-- The following program is for testing the code
-- If want to use some part of the code, just uncomment it
-------------------------------------------------------------------------------------------------------------------------
-- x -> x
-- datagen1 : process
-- Since we're going to output a number of different test
-- data we need an internal counter to keep track of things.
--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;

-----------------------------------------------------------------------------------------------------------------------

--x->y

--datagen2 : 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 <= "00000011"; -- 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 <= "10101110"; -- 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 datagen2;

-----------------------------------------------------------------------------------------------------------------------

--y->y

--datagen3 : 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

--stim_yavail <= '1'; -- data flag raised

--wait until internalclock'event and internalclock = '1'

--and stim_yenable = '1';

--stim_ydata <= "00000011"; -- header word pushed when system

--stim_yavail <= '0'; -- requests transfer, and flag back down

--wait until internalclock'event and internalclock = '1'
```

```
--and stim_yenable = '1';
--stim_ydata <= "10101110"; -- data word pushed
--wait until internalclock'event and internalclock = '1'
--and stim_yenable = '1';
--stim_ydata <= "11111111"; -- end word pushed
--elsif count = 5 then -- NO OPERATION
--stim_yavail <= '0';
--stim_ydata <= "00000000";
--else -- NO OPERATION
--stim_yavail <= '0';
--stim_ydata <= "00000000";
--end if;
--end process datagen3;
-----------------------------------------------------------------------------------------------------------------------
--y->x
--datagen4 : 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
--stim_yavail <= '1'; -- data flag raised
--wait until internalclock'event and internalclock = '1'
--and stim_yenable = '1';
--stim_ydata <= "00000010"; -- header word pushed when system
--stim_yavail <= '0'; -- requests transfer, and flag back down
```

```
--wait until internalclock'event and internalclock = '1'
--and stim_yenable = '1';
--stim_ydata <= "10101110"; -- data word pushed
--wait until internalclock'event and internalclock = '1'
--and stim_yenable = '1';
--stim_ydata <= "11111111"; -- end word pushed
--elsif count = 5 then -- NO OPERATION
--stim_yavail <= '0';
--stim_ydata <= "00000000";
--else -- NO OPERATION
--stim_yavail <= '0';
--stim_ydata <= "00000000";
--end if;
--end process datagen4;
-- ************************************************************************************************
-- 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
-- Set the default state and outputs.
present_state <= poll_x;
gntx <= '0';
gnty <= '0';
```

```
xdata_out <= "00000000";
ydata_out <= "00000000";
elsif (clk'event and clk = '1') then
-- Set the default state and outputs.
present_state <= poll_x;
gntx <= '0';
gnty <= '0';
xdata_out <= "00000000";
ydata_out <= "00000000";
case present_state is
when poll_x =>
if (x_reqx = '1' or y_reqx = '1') then
present_state <= grant_x;
else
present_state <= poll_y;
end if;
when poll_y =>
if (x_reqx = '1' or y_reqx = '1') then
present_state <= grant_y;
else
present_state <= poll_x;
end if;
when grant_x =>
if (y_writex = '1') then
xdata_out <= stim_ydata;
elsif (writex = '1') then
xdata_out <=stim_xdata;
end if;
```

```
if (x_reqx = '1') then
present_state <= grant_x;
elsif(y_reqx = '1') then
present_state <= grant_x;
else
present_state <= poll_y;
end if;
gntx <= '1';
when grant_y =>
if (x_writey = '1') then
ydata_out <= stim_xdata;
elsif (writey = '1') then
ydata_out <= stim_ydata;
end if;
if (x_reqy = '1') then
present_state <= grant_y;
elsif(y_reqy= '1') then
present_state <= grant_y;
else
present_state <= poll_x;
end if;
gntx <= '1';
when others =>
-- Set the default state and outputs.
present_state <= poll_x;
gntx <= '0';
gnty <= '0';
xdata_out <= "00000000";
```

```
ydata_out <= "00000000";

end case;

end if;

end process;

end behavioral;
```

end of top_tb.vhd