Glasgow College, UESTC

DIGITAL CIRCUIT DESIGN FINAL REPORT

Crossbar Switch Design and Test

Name: Zhang Licheng

UoG ID: 2357767

UESTC ID: 2017200602011

Date: 20. 12. 2019

Introduction

The aim of this laboratory is designing a crossbar switch system to route arbitrary length 8-bit word data packets by using Vivado to gain experience in solving a realistic digital design. For the crossbar switch, data from the x/y-input can flow to the x and/or y outputs, which means there are single-channel, dual-channel and broadcast operation cases. By VHDL coding, I designed and tested all cases by simulating in Vivado. And my design of the state machines, simulation results, relevant analysis and code implementation are all included in this final laboratory report.

Design and Analysis

Data from the x-input can flow to the x and/or y outputs. And 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.

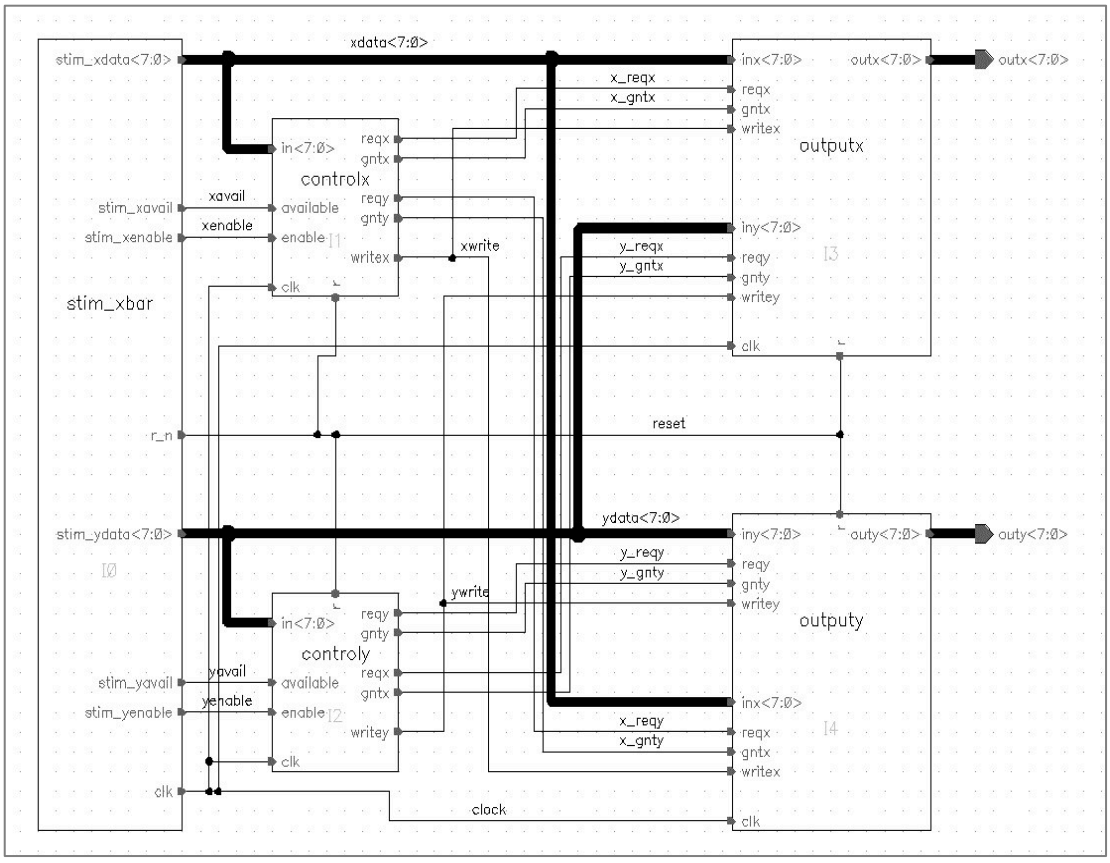

Figure1. Schematic of Crossbar Switch

There should be eight kinds of different operations completed by the designed crossbar switch.

Case1: X -> X & Y -> Y (Parallel dual-channel data transmission);

Case2: X -> Y & Y -> X (Cross dual-channel data transmission);

Case3: X -> X & X -> Y (Broadcast data transmission);

Case4: Y -> X & Y -> X (Broadcast data transmission);

Case5: X -> X & Nothing -> Y (Single-channel data transmission);

Case6: Y -> Y & Nothing -> X (Single-channel data transmission);

Case7: X -> Y & Nothing -> X (Single-channel data transmission);

Case8: Y -> X & Nothing -> Y (Single-channel data transmission).

In practice, it is easy to complete four operations by setting LSB with 1 or 0 but it is complex to complete all these eight operations by designing only one crossbar switch, because setting LSB does not work for eight operations. Finally, I figured out a new method, that is, setting Last three bits instead of LSB with 010, 011,100 and 110 so that the input port and output can be determined.

And I constructed state machines for top_tb(in *Figure2*) and control(in *Figure3*) modules with what I have learnt from Digital Circuit Design lectures. Then I translated them into VHDL for operation tests.

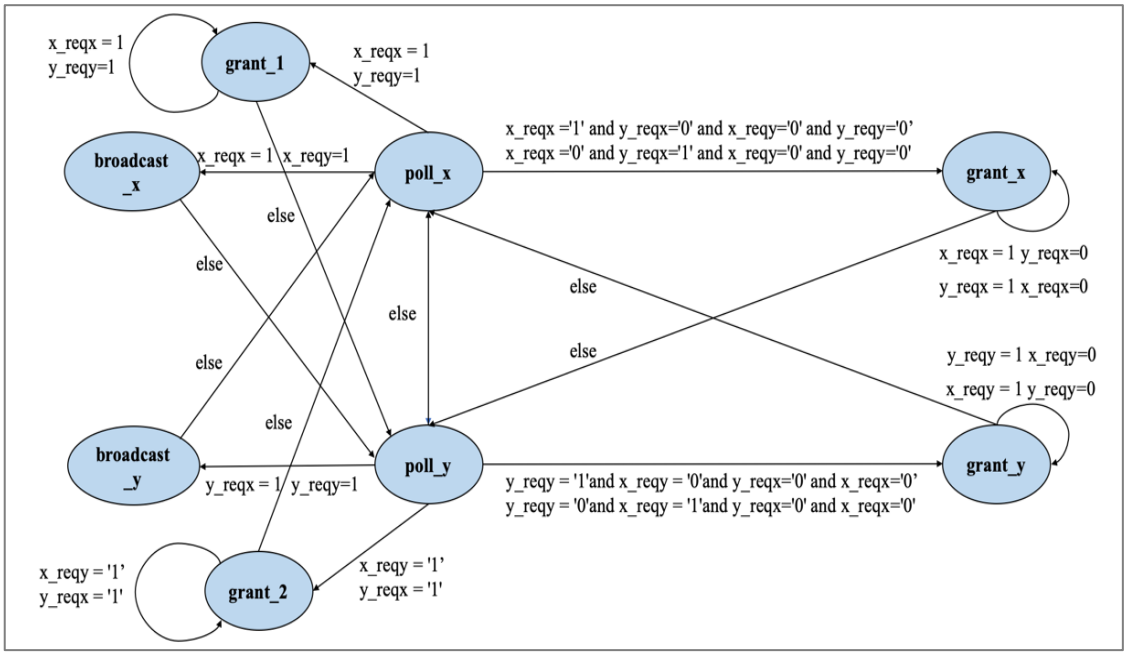

Figure2. State machine of top_tb module

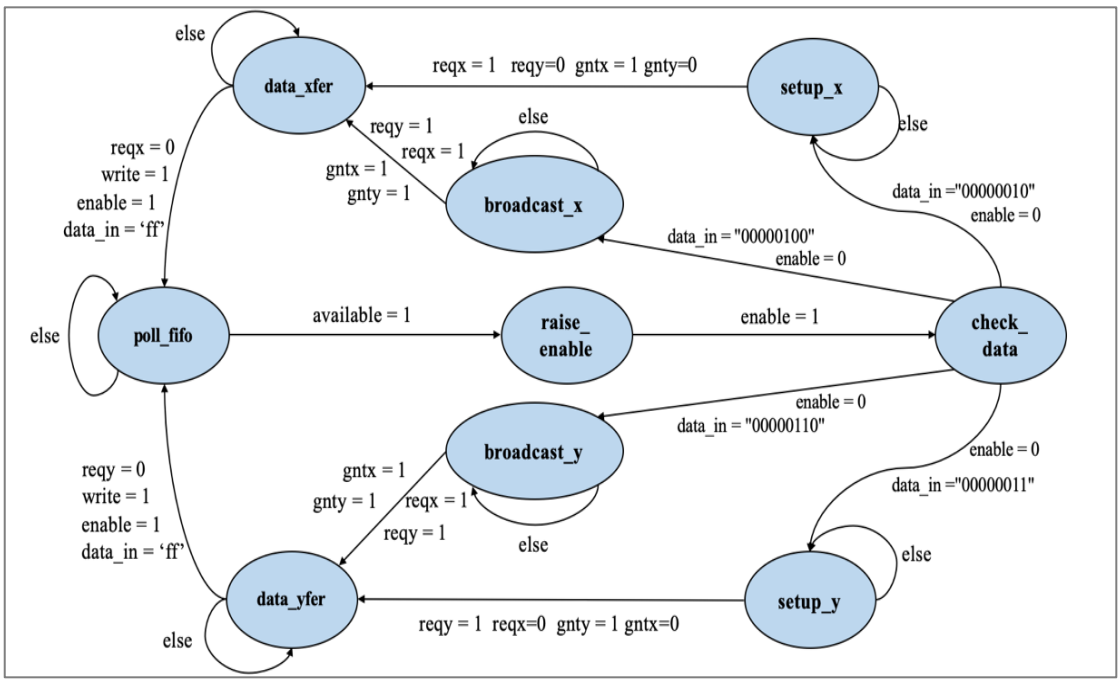

Figure3. State machine of control module

After coding by VHDL in Vivado, top_tb and control modules are completed. By using the simulation tools of the CAD design suite, I obtained the schematic of control module(in *Figure4*) which describe the structure and principle of it clearly.

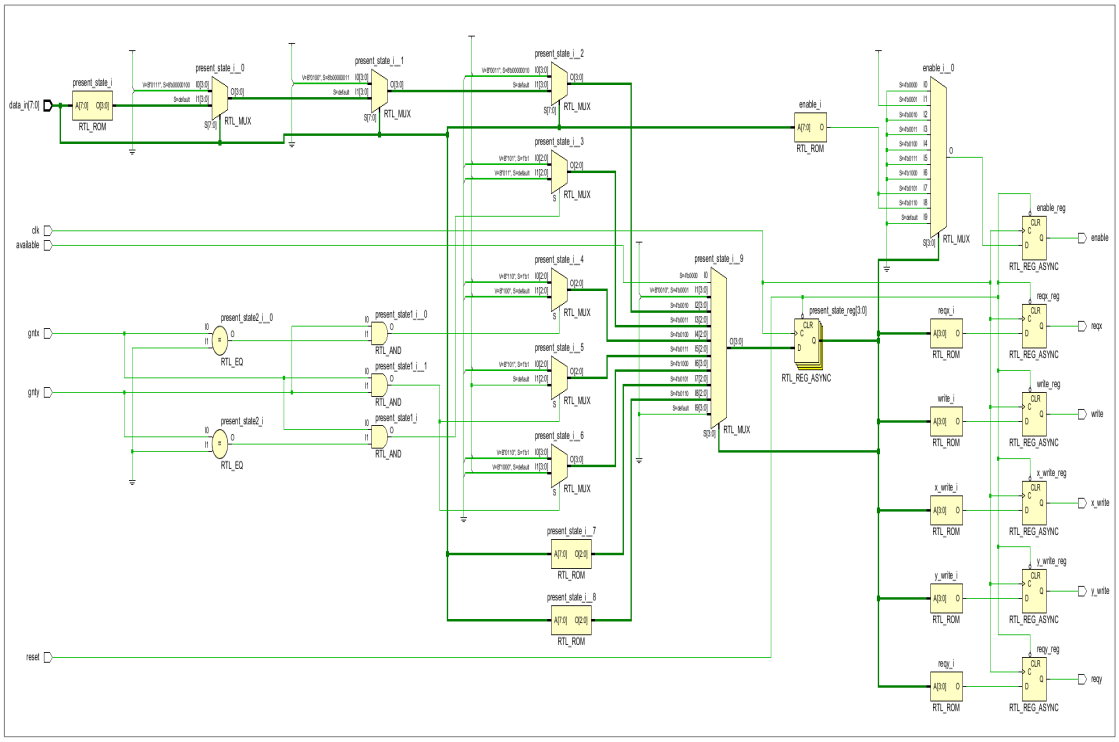

Figure4. Schematic of control module

Test and Result

To test the operation of the crossbar switch, I developed a comprehensive set of test vectors. And I created a table as shown in *table1,* where a brief analysis of simulation results in terms of the input ports, output ports, transferred data and received data are included. It also indicates that the crossbar switch designed is functionally correct in terms of my simulation results.

Case	Simulation Result	Type	Input port	Output port	Transferred data	Received data
1	*Figure5*	Parallel Dual-channel	X Y	X Y	e6 ae	e6 ae
2	*Figure6*	Cross Dual-channel	X Y	Y X	e6 ae	e6 ae
3	*Figure7*	Broadcast	X	X Y	e6 e6	e6 e6
4	*Figure8*	Broadcast	Y	X Y	ae ae	ae ae
5	*Figure9*	Single-channel	X	X	e6	e6
6	*Figure10*	Single-channel	Y	Y	e6	e6
7	*Figure11*	Single-channel	X	Y	e6	e6
8	*Figure12*	Single-channel	Y	X	e6	e6

Table1. Simulation result

As can be seen from *table1*, there are eight kind of different operations completed by my crossbar switch, and simulation results are shown from Case1(in *Figure5)* to Case8(in *Figure12)* as below, respectively.

Case1: X-input to X-output & Y-input to Y-output

As can be seen from the simulation result below, it is a parallel dual-channel transmission, where data(e6) is transferred from X-input to X-output and data(ae) is transferred from Y-input to Y-output simultaneously. The transmission ends up with ff.

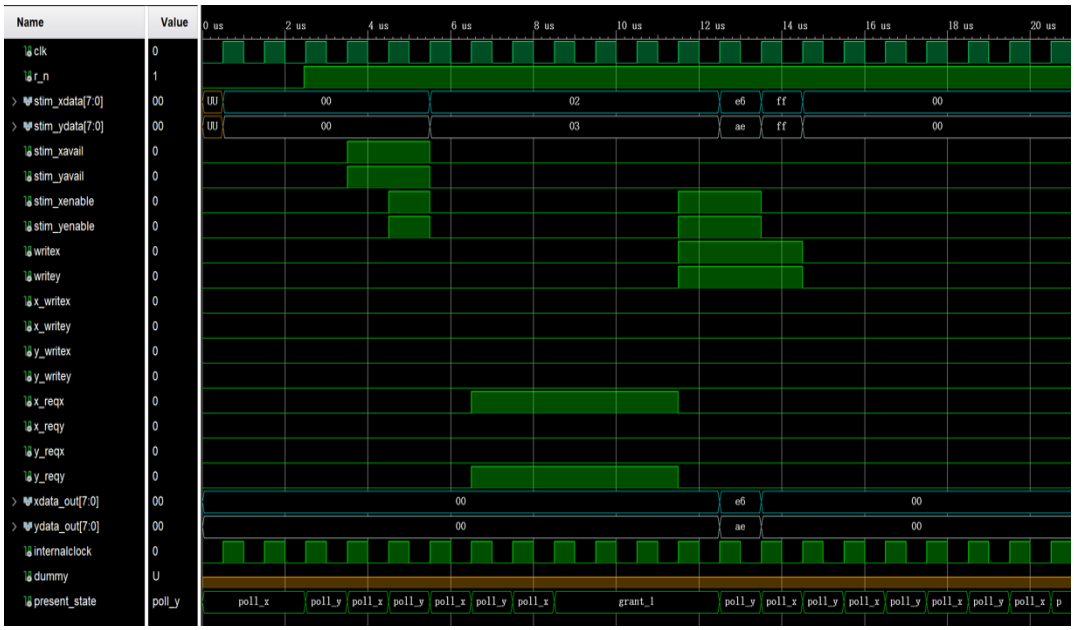

Figure5

Case2: X-input to Y-output & Y-input to X-output

As can be seen from the simulation result below, it is a cross dual-channel transmission, where data(e6) is transferred from X-input to Y-output and data(ae) is transferred from Y-input to X-output simultaneously. The transmission ends up with ff.

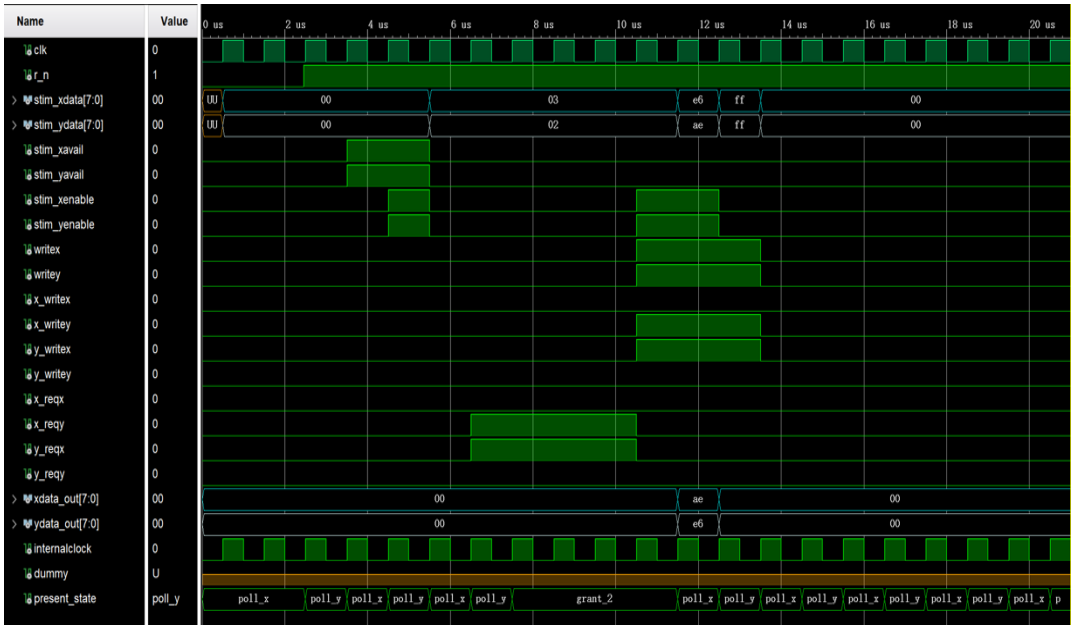

Figure6

Case3: X-input to X-output & X-input to Y-output

As can be seen from the simulation result below, it is a broadcast transmission, where data(e6) is transferred from X-input to X-output and Y-output simultaneously. The transmission ends up with ff.

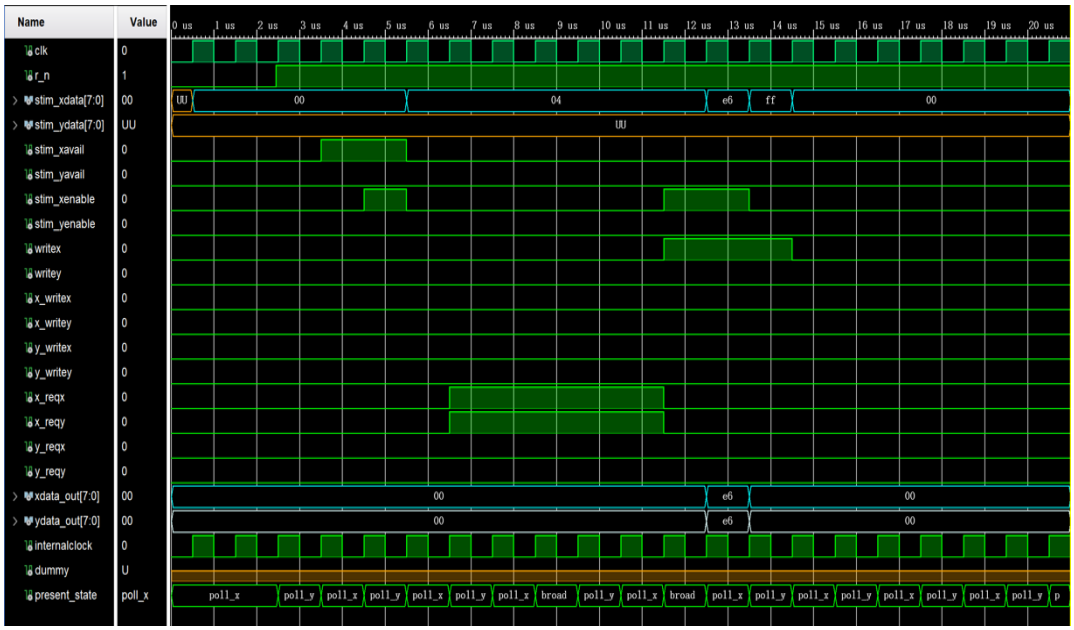

Figure7

Case4: Y-input to X-output & Y-input to Y-output

As can be seen from the simulation result below, it is a variant of broadcast transmission in case3, where the input is Y rather than X. Data(ae) is transferred from Y-input to X-output and Y-output simultaneously. The transmission ends up with ff.

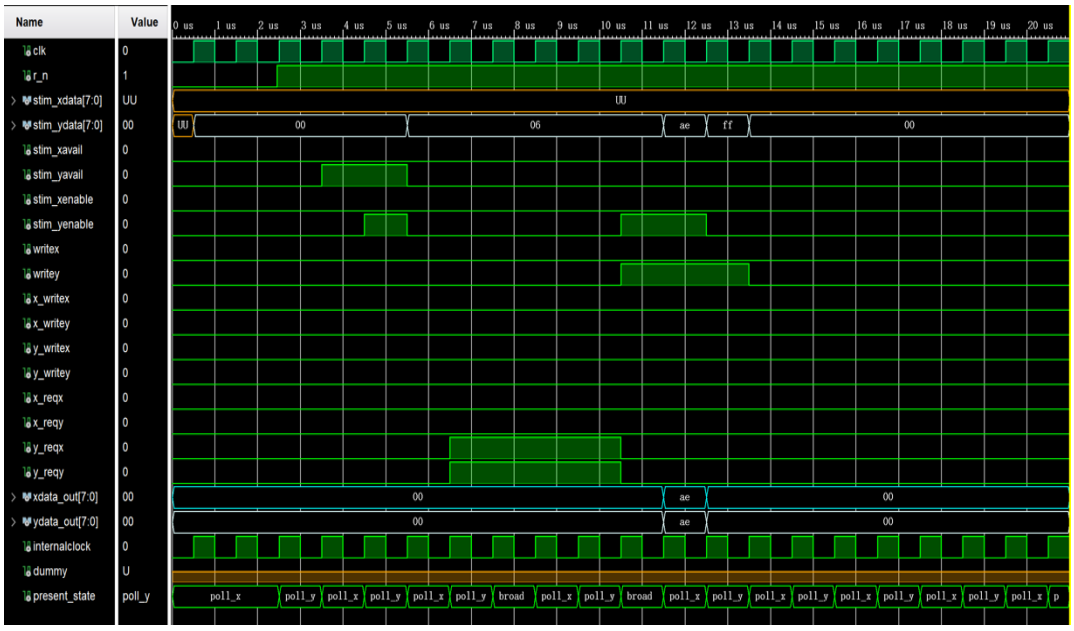

Figure8

Case5: X-input to X-output & Nothing to Y-output

As can be seen from the simulation result below, it is a simple single-channel transmission, where only one data(e6) is transferred from X-input to X-output. The transmission ends up with ff.

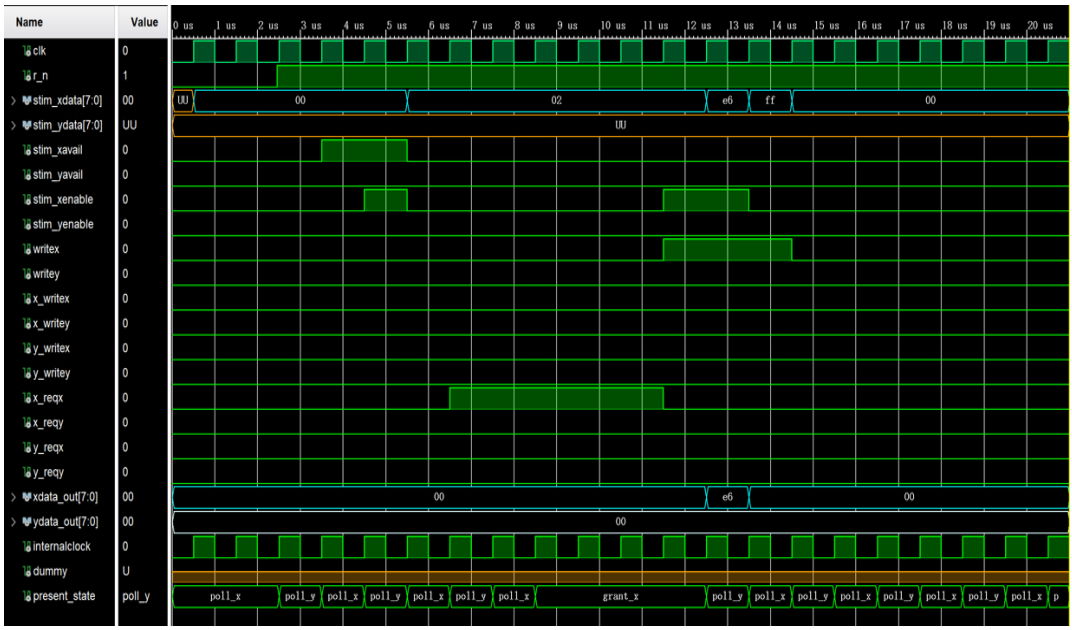

Figure9

Case6: Y-input to Y-output & Nothing to X-output

As can be seen from the simulation result below, it is similar to the single-channel transmission in case5, where the input is Y rather than X. Only one data(e6) is transferred from Y-input to Y-output. The transmission ends up with ff.

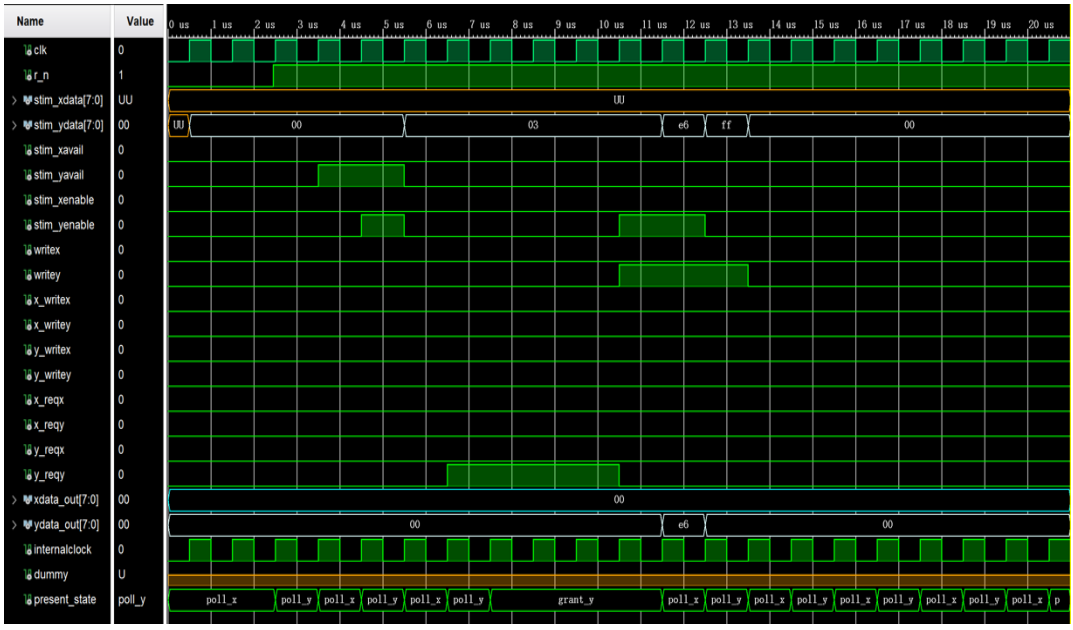

Figure10

Case7: X-input to Y-output & Nothing to X-output

As can be seen from the simulation result below, it is a simple single-channel transmission, where only one data(e6) is transferred from X-input to Y-output. The transmission ends up with ff.

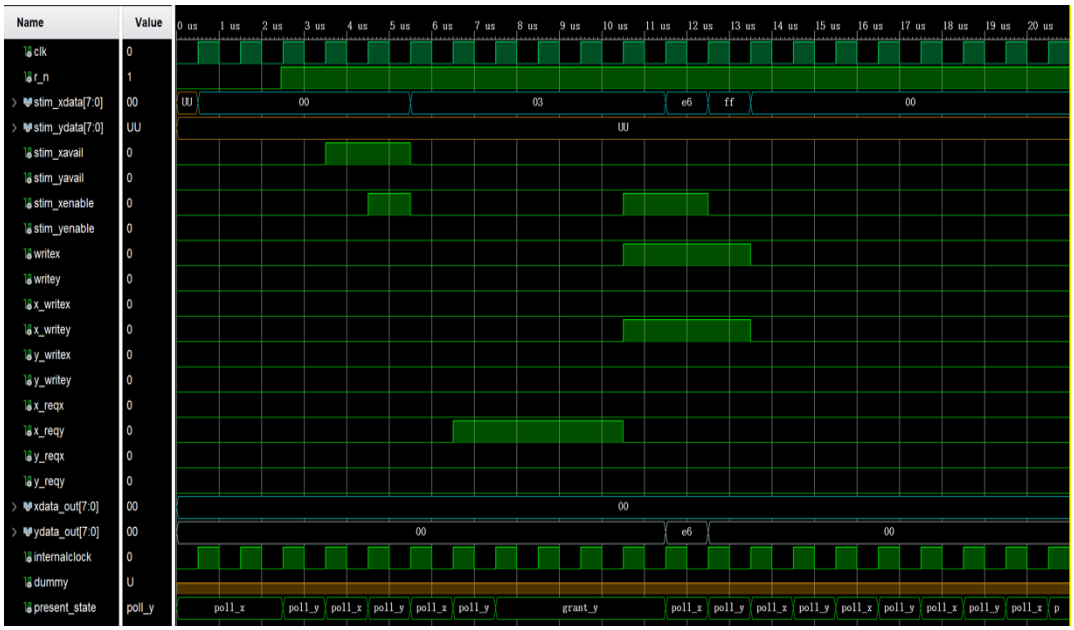

Figure11

Case8: Y-input to X-output & Nothing to Y-output

As can be seen from the simulation result below, it is similar to the single-channel transmission in case7, where the input is Y rather than X. Only one data(e6) is transferred from Y-input to X-output. The transmission ends up with ff.

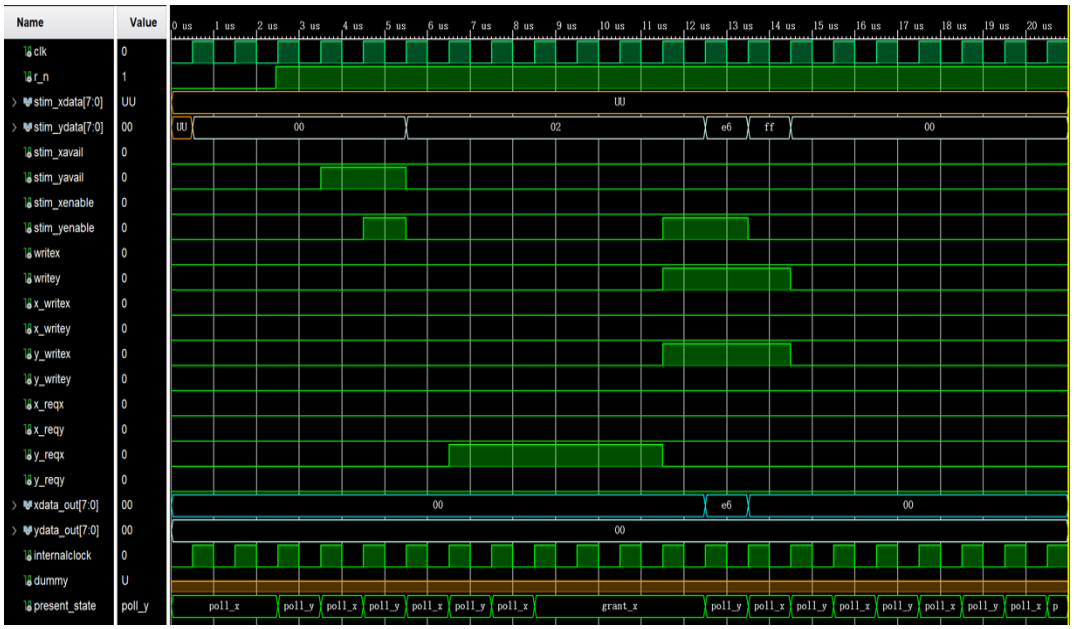

Figure12

Conclusion

During this experiment, I have learned how to design a crossbar switch which can transfer data through single-channel or dual-channel by VHDL. After constructing and translating the state machine for controlx/y into VHDL, the completed module is tested by using the simulation tools and test vectors described in top_tb.vhd. There are 8 operation cases of the crossbar switch and each of them has been tested successfully.

Overall, the crossbar switch designed by myself is functionally correct, easy to synthesize, and retains maintainability based on my simulation results.

Reference

[1] M. Morris Mano, Michael D. Ciletti, *Digital Design with an Introduction to the Verilog HDL*,5th ed, Pearson Eduction, 2013;

[2] Peter J. Ashenden, *The VHDL Cookbook*, 1st ed, 1990.

Appendix

The code for top_tb.vhd:

```
1.  --------------------------------------------------------------------------------
2.  -- Company: Glasgow College, UESTC
3.  -- Engineer: ZHANG Licheng
4.  -- Create Date: 2019/11/07 20:35:12
5.  -- Design Name: LAB3
6.  -- Module Name: LAB3 – top_tb
7.  -- Project Name: Crossbar Switch Design
8.  -- Revision:
9.  -- Revision 0.01 - File Created
10. --------------------------------------------------------------------------------
11. library IEEE;
12. use IEEE.STD_LOGIC_1164.ALL;
13. use IEEE.NUMERIC_STD.all; -- for internal counter etc.
14. -- Uncomment the following library declaration if using
15. -- arithmetic functions with Signed or Unsigned values
16. --use IEEE.NUMERIC_STD.ALL;
17.
```

```
18. -- Uncomment the following library declaration if instantiating
19. -- any Xilinx leaf cells in this code.
20. --library UNISIM;
21. --use UNISIM.VComponents.all;
22. 
23. entity top_tb is
24. --  Port ( );
25. end top_tb;
26. 
27. architecture behavioral of top_tb is
28. component control
29. PORT(
30. enable : OUT std_logic;
31. reqx : OUT std_logic;
32. reqy : OUT std_logic;
33. write : OUT std_logic;
34. x_write : OUT std_logic;
35. y_write : OUT std_logic;
36. available : IN std_logic;
37. clk : IN std_logic;
38. data_in : IN std_logic_vector(7 DOWNTO 0);
39. gntx : IN std_logic;
40. gnty : IN std_logic;
41. reset : IN std_logic
42. );
43. end component;
44. --INPUT
45. signal clk: std_logic;
46. signal r_n: std_logic;
47. signal stim_xdata: std_logic_vector(7 downto 0);
48. signal stim_ydata: std_logic_vector(7 downto 0);
49. signal stim_xavail: std_logic :='0';
50. signal stim_yavail: std_logic :='0';
51. signal x_gntx: std_logic :='0';
52. signal x_gnty: std_logic :='0';
53. signal y_gntx: std_logic :='0';
54. signal y_gnty: std_logic :='0';
55. --OUTPUT
56. signal stim_xenable: std_logic :='0';
57. signal stim_yenable: std_logic :='0';
58. signal writex: std_logic :='0';
59. signal writey: std_logic :='0';
60. signal x_writex: std_logic :='0';
61. 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);
signal xdata_out: std_logic_vector(7 downto 0);
signal ydata_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_1, grant_2, grant_x,grant_y,broadcast_
x, broadcast_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=>x_gntx,
gnty=>x_gnty,
--Output
enable=>stim_xenable,
write=>writex,
x_write=>x_writey,
reqx=>x_reqx,
reqy=>x_reqy
);
controly:control port map(
--Input
clk=>clk,
reset=>r_n,
data_in=>stim_ydata,
available=>stim_yavail,
gntx=>y_gntx,
gnty=>y_gnty,
--Output
```

```
105. enable=>stim_yenable,
106. write=>writey,
107. y_write=>y_writex,
108. reqx=>y_reqx,
109. reqy=>y_reqy
110. );
111. resetgen : process
112. begin -- process resetgen
113. r_n <= '0';
114. wait for 2450 ns;
115. r_n <= '1';
116. -- Add more entries if required at this point !
117. wait until dummy'event;
118. end process resetgen;
119. -- purpose: Generates the internal clock source. This is a 50% duty
120. -- cycle clock source, period 1000ns, with the first half of
121. -- the cycle being logic '0'.
122. --
123. -- outputs: internalclock
124. clockgen : process
125. begin -- process clockgen
126. internalclock <= '0';
127. wait for 500 ns;
128. internalclock <= '1';
129. wait for 500 ns;
130. end process clockgen;
131. -- purpose: Generates the stimuli for the data, using a counter based
132. -- approach which is much cleaner than explicit timings
133. --
134. -- outputs: stim_inst
135. -- stim_a
136. -- stim_b
137.
138. -- ****************************************************************
139. --All possible cases:
140. --Case1 X -> X & Y -> Y (Parallel dual-channel data transmission);
141. --Case2 X -> Y & Y -> X (Cross dual-channel data transmission);
142. --Case3 X -> X & X -> Y (Broadcast data transmission);
143. --Case4 Y -> X & Y -> X (Broadcast data transmission);
144. --Case5 X -> X & Nothing -> Y (Single-channel data transmission);
145. --Case6 Y -> Y & Nothing -> X (Single-channel data transmission);
146. --Case7 X -> Y & Nothing -> X (Single-channel data transmission);
147. --Case8 Y -> X & Nothing -> Y (Single-channel data transmission);
148. --Codes used to simulate these eight cases are shown as below respectively
```

```
149. --For each case, only the corresponding part of codes should be included
150. -- ****************************************************************
151.
152. -- ***********************************
153. --Case1 X -> X & Y -> Y;
154. --x -> x
155. datagen1 : process
156. -- Since we're going to output a number of different test
157. -- data we need an internal counter to keep track of things.
158. variable count : unsigned (5 downto 0) := "000000";
159. begin -- process datagen1
160. wait until internalclock'event and internalclock = '1' ;
161. count := count + 1;
162. if count = 4 then -- simple data transfer x -> x
163. stim_xavail <= '1';  -- data flag raised
164. wait until internalclock'event and internalclock = '1'
165. and stim_xenable = '1';
166. stim_xdata <= "00000010"; -- header word pushed when system
167. stim_xavail <= '0'; -- requests transfer, and flag back down
168. wait until internalclock'event and internalclock = '1'
169. and stim_xenable = '1';
170. stim_xdata <= "11100110"; -- data word pushed
171. wait until internalclock'event and internalclock = '1'
172. and stim_xenable = '1';
173. stim_xdata <= "11111111"; -- end word pushed
174. elsif count = 5 then -- NO OPERATION
175. stim_xavail <= '0';
176. stim_xdata <= "00000000";
177. else -- NO OPERATION
178. stim_xavail <= '0';
179. stim_xdata <= "00000000";
180. end if;
181. end process datagen1;
182. --y -> y
183. Datagen3 : process
184. -- Since we're going to output a number of different test
185. -- data we need an internal counter to keep track of things.
186. variable count : unsigned (5 downto 0) := "000000";
187. begin -- process datagen1
188. wait until internalclock'event and internalclock = '1';
189. count := count + 1;
190. if count = 4 then -- simple data transfer x -> x
191. stim_yavail <= '1';  -- data flag raised
192. wait until internalclock'event and internalclock = '1'
```

```
193. and stim_yenable = '1';
194. stim_ydata <= "00000011"; -- header word pushed when system
195. stim_yavail <= '0'; -- requests transfer, and flag back down
196. wait until internalclock'event and internalclock = '1'
197. and stim_yenable = '1';
198. stim_ydata <= "10101110"; -- data word pushed
199. wait until internalclock'event and internalclock = '1'
200. and stim_yenable = '1';
201. stim_ydata <= "11111111"; -- end word pushed
202. elsif count = 5 then -- NO OPERATION
203. stim_yavail <= '0';
204. stim_ydata <= "00000000";
205. else -- NO OPERATION
206. stim_yavail <= '0';
207. stim_ydata <= "00000000";
208. end if;
209. end process datagen3;
210. -- ************************************
211.
212. -- ************************************
213. --Case2 X -> Y & Y -> X;
214. -- x -> y
215. datagen2 : process
216. -- Since we're going to output a number of different test
217. -- data we need an internal counter to keep track of things.
218. variable count : unsigned (5 downto 0) := "000000";
219. begin -- process datagen2
220. wait until internalclock'event and internalclock = '1';
221. count := count + 1;
222. if count = 4 then -- simple data transfer x -> y
223. stim_xavail <= '1'; -- data flag raised
224. wait until internalclock'event and internalclock = '1'
225. and stim_xenable = '1';
226. stim_xdata <= "00000011"; -- header word pushed when system
227. stim_xavail <= '0'; -- requests transfer, and flag back down
228. wait until internalclock'event and internalclock = '1'
229. and stim_xenable = '1';
230. stim_xdata <= "11100110"; -- data word pushed
231. wait until internalclock'event and internalclock = '1'
232. and stim_xenable = '1';
233. stim_xdata <= "11111111"; -- end word pushed
234. elsif count = 5 then -- NO OPERATION
235. stim_xavail <= '0';
236. stim_xdata <= "00000000";
```

```
237. else -- NO OPERATION
238. stim_xavail <= '0';
239. stim_xdata <= "00000000";
240. end if;
241. end process datagen2;
242. -- y -> x
243. datagen4 : process
244. -- Since we're going to output a number of different test
245. -- data we need an internal counter to keep track of things.
246. variable count : unsigned (5 downto 0) := "000000";
247. begin -- process datagen4
248. wait until internalclock'event and internalclock = '1';
249. count := count + 1;
250. if count = 4 then -- simple data transfer y ->x
251. stim_yavail <= '1'; -- data flag raised
252. wait until internalclock'event and internalclock = '1'
253. and stim_yenable = '1';
254. stim_ydata <= "00000010"; -- header word pushed when system
255. stim_yavail <= '0'; -- requests transfer, and flag back down
256. wait until internalclock'event and internalclock = '1'
257. and stim_yenable = '1';
258. stim_ydata <= "10101110"; -- data word pushed
259. wait until internalclock'event and internalclock = '1'
260. and stim_yenable = '1';
261. stim_ydata <= "11111111"; -- end word pushed
262. elsif count = 5 then -- NO OPERATION
263. stim_yavail <= '0';
264. stim_ydata <= "00000000";
265. else -- NO OPERATION
266. stim_yavail <= '0';
267. stim_ydata <= "00000000";
268. end if;
269. end process datagen4;
270. -- ************************************
271.
272. -- ************************************
273. --Case3 X -> X & X -> Y;
274. --x -> x & x -> y
275. datagen5 : process
276. -- Since we're going to output a number of different test
277. -- data we need an internal counter to keep track of things.
278. variable count : unsigned (5 downto 0) := "000000";
279. begin -- process datagen1
280. wait until internalclock'event and internalclock = '1';
```

```
281. count := count + 1;
282. if count = 4 then -- simple data transfer x -> x
283. stim_xavail <= '1'; -- data flag raised
284. wait until internalclock'event and internalclock = '1'
285. and stim_xenable = '1';
286. stim_xdata <= "00000100"; -- header word pushed when system
287. stim_xavail <= '0'; -- requests transfer, and flag back down
288. wait until internalclock'event and internalclock = '1'
289. and stim_xenable = '1';
290. stim_xdata <= "11100110"; -- data word pushed
291. wait until internalclock'event and internalclock = '1'
292. and stim_xenable = '1';
293. stim_xdata <= "11111111"; -- end word pushed
294. elsif count = 5 then -- NO OPERATION
295. stim_xavail <= '0';
296. stim_xdata <= "00000000";
297. else -- NO OPERATION
298. stim_xavail <= '0';
299. stim_xdata <= "00000000";
300. end if;
301. end process datagen5;
302. -- ************************************
303.
304. -- ************************************
305. --Case4 Y -> X & Y -> Y;
306. --y -> x & y -> y
307. Datagen6 : process
308. -- Since we're going to output a number of different test
309. -- data we need an internal counter to keep track of things.
310. variable count : unsigned (5 downto 0) := "000000";
311. begin -- process datagen1
312. wait until internalclock'event and internalclock = '1';
313. count := count + 1;
314. if count = 4 then
315. stim_yavail <= '1'; -- data flag raised
316. wait until internalclock'event and internalclock = '1'
317. and stim_yenable = '1';
318. stim_ydata <= "00000110"; -- header word pushed when system
319. stim_yavail <= '0'; -- requests transfer, and flag back down
320. wait until internalclock'event and internalclock = '1'
321. and stim_yenable = '1';
322. stim_ydata <= "10101110"; -- data word pushed
323. wait until internalclock'event and internalclock = '1'
324. and stim_yenable = '1';
```

```
325. stim_ydata <= "11111111"; -- end word pushed
326. elsif count = 5 then -- NO OPERATION
327. stim_yavail <= '0';
328. stim_ydata <= "00000000";
329. else -- NO OPERATION
330. stim_yavail <= '0';
331. stim_ydata <= "00000000";
332. end if;
333. end process datagen6;
334. -- ************************************
335.
336. -- ************************************
337. --Case5 X -> X & Nothing -> Y;
338. --x -> x
339. datagen1 : process
340. -- Since we're going to output a number of different test
341. -- data we need an internal counter to keep track of things.
342. variable count : unsigned (5 downto 0) := "000000";
343. begin -- process datagen1
344. wait until internalclock'event and internalclock = '1';
345. count := count + 1;
346. if count = 4 then -- simple data transfer x -> x
347. stim_xavail <= '1'; -- data flag raised
348. wait until internalclock'event and internalclock = '1'
349. and stim_xenable = '1';
350. stim_xdata <= "00000010"; -- header word pushed when system
351. stim_xavail <= '0'; -- requests transfer, and flag back down
352. wait until internalclock'event and internalclock = '1'
353. and stim_xenable = '1';
354. stim_xdata <= "11100110"; -- data word pushed
355. wait until internalclock'event and internalclock = '1'
356. and stim_xenable = '1';
357. stim_xdata <= "11111111"; -- end word pushed
358. elsif count = 5 then -- NO OPERATION
359. stim_xavail <= '0';
360. stim_xdata <= "00000000";
361. else -- NO OPERATION
362. stim_xavail <= '0';
363. stim_xdata <= "00000000";
364. end if;
365. end process datagen1;
366. -- ************************************
367.
368. -- ************************************
```

```
369. --Case6 Y -> Y & Nothing -> X;
370. -- y -> y
371. datagen3 : process
372. -- Since we're going to output a number of different test
373. -- data we need an internal counter to keep track of things.
374. variable count : unsigned (5 downto 0) := "000000";
375. begin -- process datagen3
376. wait until internalclock'event and internalclock = '1';
377. count := count + 1;
378. if count = 4 then -- simple data transfer y -> y
379. stim_yavail <= '1'; -- data flag raised
380. wait until internalclock'event and internalclock = '1'
381. and stim_yenable = '1';
382. stim_ydata <= "00000011"; -- header word pushed when system
383. stim_yavail <= '0'; -- requests transfer, and flag back down
384. wait until internalclock'event and internalclock = '1'
385. and stim_yenable = '1';
386. stim_ydata <= "11100110"; -- data word pushed
387. wait until internalclock'event and internalclock = '1'
388. and stim_yenable = '1';
389. stim_ydata <= "11111111"; -- end word pushed
390. elsif count = 5 then -- NO OPERATION
391. stim_yavail <= '0';
392. stim_ydata <= "00000000";
393. else -- NO OPERATION
394. stim_yavail <= '0';
395. stim_ydata <= "00000000";
396. end if;
397. end process datagen3;
398. -- ************************************
399.
400. -- ************************************
401. --Case7 X -> Y & Nothing -> X;
402. -- x -> y
403. datagen2 : process
404. -- Since we're going to output a number of different test
405. -- data we need an internal counter to keep track of things.
406. variable count : unsigned (5 downto 0) := "000000";
407. begin -- process datagen2
408. wait until internalclock'event and internalclock = '1';
409. count := count + 1;
410. if count = 4 then -- simple data transfer x -> y
411. stim_xavail <= '1'; -- data flag raised
412. 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 <= "11100110"; -- 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;
-- ************************************

-- ************************************
--Case8 Y -> X & Nothing -> Y;
-- 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 datagen4
wait until internalclock'event and internalclock = '1';
count := count + 1;
if count = 4 then -- simple data transfer y -> x
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 <= "11100110"; -- 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";
```

```
457. else -- NO OPERATION
458. stim_yavail <= '0';
459. stim_ydata <= "00000000";
460. end if;
461. end process datagen4;
462. -- ************************************
463.
464. clk <= internalclock;
465. --output.vhd
466. process (clk, r_n)
467. begin
468. -- Activities triggered by asynchronous reset (active low).
469. if (r_n = '0') then
470. -- Set the default state and outputs.
471. present_state <= poll_x;
472. x_gntx <= '0';
473. x_gnty <= '0';
474. xdata_out <= "00000000";
475. ydata_out <= "00000000";
476. elsif (clk'event and clk = '1') then
477. -- Set the default state and outputs.
478. present_state <= poll_x;
479. x_gnty <= '0';
480. y_gnty <= '0';
481. xdata_out <= "00000000";
482. ydata_out <= "00000000";
483. case present_state is
484.
485. when poll_x =>
486. if(x_reqx = '1' and x_reqy='1') then
487. present_state <= broadcast_x;
488. elsif (x_reqx = '1' and y_reqy='1') then
489. present_state <= grant_1;
490. elsif (x_reqx ='1' and y_reqx='0' and x_reqy='0' and y_reqy='0')then
491. present_state <= grant_x;
492. elsif (x_reqx ='0' and y_reqx='1' and x_reqy='0' and y_reqy='0')then
493. present_state <= grant_x;
494. else
495. present_state <= poll_y;
496. end if;
497.
498. when poll_y =>
499. if(y_reqx = '1' and y_reqy='1') then
500. present_state <= broadcast_y;
```

```
elsif (x_reqy = '1'and y_reqx = '1') then
present_state <= grant_2;
elsif (y_reqy = '1'and x_reqy = '0'and y_reqx='0' and x_reqx='0') then
present_state <= grant_y;
elsif (y_reqy = '0'and x_reqy = '1'and y_reqx='0' and x_reqx='0') then
present_state <= grant_y;
else
present_state <= poll_x;
end if;

when broadcast_x =>
if (writex = '1') then
xdata_out <= stim_xdata;
ydata_out <= stim_xdata;
elsif (x_reqx = '1' and y_reqx='1') then
present_state <= broadcast_x;
else
present_state <= poll_y;
end if;
x_gntx <= '1';
y_gntx <= '0';
x_gnty <= '1';
y_gnty <= '0';

when broadcast_y =>
if (writey = '1') then
xdata_out <= stim_ydata;
ydata_out <= stim_ydata;
elsif (x_reqy = '1' and y_reqy='1') then
present_state <= broadcast_y;
else
present_state <= poll_x;
end if;
x_gntx <= '0';
y_gntx <= '1';
x_gnty <= '0';
y_gnty <= '1';

when grant_1 =>
if (writex = '1'and writey = '1') then
xdata_out <= stim_xdata;
ydata_out <= stim_ydata;
end if;
if (x_reqx = '1'and y_reqy = '1') then
```

```
545. present_state <= grant_1;
546. else
547. present_state <= poll_y;
548. end if;
549. x_gntx <= '1';
550. y_gntx <= '0';
551. x_gnty <= '0';
552. y_gnty <= '1';
553.
554. when grant_2 =>
555. if (x_writey = '1'and y_writex = '1') then
556. ydata_out <= stim_xdata;
557. xdata_out <= stim_ydata;
558. end if;
559. if (y_reqx = '1'and x_reqy = '1') then
560. present_state <= grant_2;
561. else
562. present_state <= poll_x;
563. end if;
564. x_gntx <= '0';
565. y_gntx <= '1';
566. x_gnty <= '1';
567. y_gnty <= '0';
568.
569. when grant_x =>
570. if (y_writex = '1') then
571. xdata_out <= stim_ydata;
572. elsif (writex = '1') then
573. xdata_out <= stim_xdata;
574. end if;
575. if (x_reqx = '1'and y_reqx='0') then
576. present_state <= grant_x;
577. x_gntx <= '1';
578. y_gntx <= '0';
579. x_gnty <= '0';
580. y_gnty <= '0';
581. elsif(y_reqx = '1'and x_reqx ='0') then
582. present_state <= grant_x;
583. x_gntx <= '0';
584. y_gntx <= '1';
585. x_gnty <= '0';
586. y_gnty <= '0';
587. else
588. present_state <= poll_y;
```

```
589. end if;
590.
591. when grant_y =>
592. if (x_writey = '1') then
593. ydata_out <= stim_xdata;
594. elsif (writey = '1') then
595. ydata_out <= stim_ydata;
596. end if;
597. if (y_reqy = '1'and x_reqy = '0') then
598. present_state <= grant_y;
599. x_gntx <= '0';
600. y_gntx <= '0';
601. x_gnty <= '0';
602. y_gnty <= '1';
603. elsif (x_reqy = '1'and y_reqy ='0') then
604. present_state <= grant_y;
605. x_gntx <= '0';
606. y_gntx <= '0';
607. x_gnty <= '1';
608. y_gnty <= '0';
609. else
610. present_state <= poll_x;
611. end if;
612.
613. when others =>
614. -- Set the default state and outputs.
615. present_state <= poll_x;
616. x_gntx <= '0';
617. y_gntx <= '0';
618. x_gnty <= '0';
619. y_gnty <= '0';
620. xdata_out <= "00000000";
621. ydata_out <= "00000000";
622. end case;
623. end if;
624. end process;
625. end Behavioral;
```

The code for control.vhd:

```
1.  ----------------------------------------------------------------------------
2.  -- Company: Glasgow College, UESTC
3.  -- Engineer: ZHANG Licheng
4.  -- Create Date: 2019/11/07 20:35:12
5.  -- Design Name: LAB3
6.  -- Module Name: LAB3 - control
7.  -- Project Name: Crossbar Switch Design
8.  -- Revision:
9.  -- Revision 0.01 - File Created
10. ----------------------------------------------------------------------------
11. library IEEE;
12. use IEEE.STD_LOGIC_1164.ALL;
13. use IEEE.NUMERIC_STD.all;
14.
15. -- Uncomment the following library declaration if using
16. -- arithmetic functions with Signed or Unsigned values
17. --use IEEE.NUMERIC_STD.ALL;
18.
19. -- Uncomment the following library declaration if instantiating
20. -- any Xilinx leaf cells in this code.
21. --library UNISIM;
22. --use UNISIM.VComponents.all;
23.
24. entity control is
25. Port (
26. enable : OUT std_logic;
27. reqx : OUT std_logic;
28. reqy : OUT std_logic;
29. write: OUT std_logic;
30. x_write : OUT std_logic;
31. y_write : OUT std_logic;
32. available : IN std_logic;
33. clk : IN std_logic;
34. data_in : IN std_logic_vector(7 DOWNTO 0);
35. gntx : IN std_logic;
36. gnty : IN std_logic;
37. reset : IN std_logic
38. );
39. end control;
40.
41. architecture Behavioral of control is
42. -- Possible states.
```

```
43. type states is (poll_fifo, raise_enable, check_data, setup_x,setup_y, data_x
    fer, data_yfer, broadcast_x, broadcast_y);
44. -- Present state.
45. signal present_state : states;
46. begin
47. -- Main process.
48. process (clk, reset)
49. begin
50. -- Activities triggered by asynchronous reset (active low).
51. if (reset = '0') then
52. -- Set the default state and outputs.
53. present_state <= poll_fifo;
54. enable <= '0';
55. reqx <= '0';
56. reqy <= '0';
57. write <= '0';
58. x_write <= '0';
59. y_write <= '0';
60. elsif (clk'event and clk = '1') then
61. -- Set the default state and outputs.
62. present_state <= poll_fifo;
63. enable <= '0';
64. reqx <= '0';
65. reqy <= '0';
66. write <= '0';
67. x_write <= '0';
68. y_write <= '0';
69. case present_state is
70.
71. when poll_fifo =>
72. if(available='1')then
73. present_state <= raise_enable;
74. else
75. present_state <= poll_fifo;
76. end if;
77.
78. when raise_enable =>
79. enable <= '1';
80. present_state <= check_data;
81.
82. when check_data =>
83. enable <= '0';
84. if(data_in ="00000010") then
85. present_state <= setup_x;
```

```
elsif(data_in = "00000011") then
present_state <= setup_y;
elsif(data_in = "00000100") then
present_state <= broadcast_x;
elsif(data_in = "00000110") then
present_state <= broadcast_y;
end if;

when setup_x =>
reqx<='1';
reqy<='0';
if(gntx='1'and gnty='0')then
present_state<=data_xfer;
else
present_state<=setup_x;
end if;

when setup_y =>
reqy<='1';
reqx<='0';
if(gnty='1'and gntx='0')then
present_state<=data_yfer;
else
present_state<=setup_y ;
end if;

when broadcast_x =>
reqx<='1';
reqy<='1';
if(gntx='1'and gnty='1')then
present_state<=data_xfer;
else
present_state<=broadcast_x;
end if;

when broadcast_y =>
reqx<='1';
reqy<='1';
if(gnty='1'and gntx='1')then
present_state<=data_yfer;
else
present_state<=broadcast_y;
end if;

```

```
when data_xfer =>
reqx<='0';
write <= '1';
y_write<='1';
enable<='1';
if(data_in = "11111111")then
enable<='0';
present_state <= poll_fifo;
else
present_state <= data_xfer;
end if;

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

when others =>
present_state <= poll_fifo;
enable <= '0';
reqx <= '0';
reqy <= '0';
x_write <= '0';
y_write <= '0';
end case;
end if;
end process;
end Behavioral;
```