

Lab7 code

```
1. //pingpong template file for Lab #7
2. //Revised by Jianjian Song to add pressing early penalty
3. //June 2019
4. #include "xparameters.h"
5. #include "xgpio.h"
6. #include "led_ip.h"
7. // Include scutimer header file
8. #include "XScuTimer.h"
9. //=====
10. XScuTimer Timer; /* Cortex A9 SCU Private Timer Instance */
11. int delay(int i);
12. void MoveBallRight(void);
13. void MoveBallLeft(void);
14.
15. #define ONE_TENTH 32500000 // half of the CPU clock speed/10
16. #define START 1
17. #define STOP 0
18. #define LEFT 0
19. #define RIGHT 1
20. #define RESETBUTTON 0b0100
21. #define STARTBUTTON 0b0010
22. #define LEFTPADDLE 0b1000
23. #define RIGHTPADDLE 0b0001
24.
25. #define LED_PATTERNS_ORDER_LEFT_OUT -1
26. #define LED_PATTERNS_ORDER_RIGHT_OUT 6
27.
28. #define RUNKEEP 1
29. #define RUNFOBBIDEN 0
30.
31. int psb_check, dip_check, LedState, Status, dip_check_prev, psb_check_prev;
32.
33. XGpio dip, push;
34.
35. // PS Timer related definitions
36. XScuTimer_Config *ConfigPtr;
37. XScuTimer *TimerInstancePtr = &Timer;
38.
39. int LED_PATTERNS[6] = { 0b0000, 0b1000, 0b0100, 0b0010, 0b0001, 0b0000 };
40. int LED_PATTERNS_ORDER[6] = {0,1,2,3,4,5};
41. int scoreright, scoreleft;
```

```

42. char GameOver, StartDirection;
43.
44. int main(void) {
45.
46.     //initialize variables, timers, ports
47.     XGpio_Initialize(&dip, XPAR_SWITCHES_DEVICE_ID);
48.     XGpio_SetDataDirection(&dip, 1, 0xffffffff);
49.
50.     XGpio_Initialize(&push, XPAR_BUTTONS_DEVICE_ID);
51.     XGpio_SetDataDirection(&push, 1, 0xffffffff);
52.
53.     //use psb_check_prev and psb_check to check if the button has been push
54.
55.     psb_check_prev = XGpio_DiscreteRead(&push, 1);
56.
57.     // Initialize the timer
58.     ConfigPtr = XScuTimer_LookupConfig(XPAR_PS7_SCUTIMER_0_DEVICE_ID);
59.     Status = XScuTimer_CfgInitialize(TimerInstancePtr, ConfigPtr,
60.                                     ConfigPtr->BaseAddr);
61.     if (Status != XST_SUCCESS) {
62.         xil_printf("Timer init() failed\r\n");
63.         return XST_FAILURE;
64.     }
65.
66.     // Read dip switch values
67.     dip_check_prev = XGpio_DiscreteRead(&dip, 1);
68.     // Load timer with delay in multiple of ONE_TENTH
69.     XScuTimer_LoadTimer(TimerInstancePtr, ONE_TENTH*dip_check_prev);
70.     // Set AutoLoad mode
71.     XScuTimer_EnableAutoReload(TimerInstancePtr);
72.     // Start the timer
73.     XScuTimer_Start(TimerInstancePtr);
74.
75.     xil_printf("-- Start of the Ping Pong Program --\r\n");
76.     GameOver = STOP;
77.     scoreright = 0;
78.     scoreleft = 0;
79.     xil_printf("Score Left = %d   Score Right = %d\r\n", scoreright, scoreleft);
80.     StartDirection = LEFT;
81.     while (1) {
82.         // Read push buttons and reset score if Button 2 is pressed
83.         psb_check = XGpio_DiscreteRead(&push, 1);

```

```

83.         if (psb_check == RESETBUTTON && psb_check != psb_check_prev)//reset
           game
84.         {
85.             xil_printf("\n\rNew Game - Scores Reset\r\n");
86.             scoreright = 0;
87.             scoreleft = 0;
88.             xil_printf("Score Left = %d   Score Right = %d\r\n", scoreright,
89.
                        scoreleft);
90.             psb_check_prev = psb_check;
91.         }
92.
93.         //check the STARTBUTTON been pushed
94.         if (psb_check == STARTBUTTON) {
95.             GameOver = START;    //start game
96.             psb_check_prev = psb_check;
97.         }
98.
99.         //check the game status,if start the ball will move right or right
100.        if (GameOver == STOP) {
101.            if (StartDirection == LEFT) {
102.                LED_IP_mWriteReg(XPAR_LED_IP_0_S00_AXI_BASEADDR, 0,
103.                                LED_PATTERNS[5]);
104.            } else {
105.                LED_IP_mWriteReg(XPAR_LED_IP_0_S00_AXI_BASEADDR, 0,
106.                                LED_PATTERNS[0]);
107.            }
108.        } else {
109.            if (StartDirection) {
110.                MoveBallLeft();
111.            } else {
112.                MoveBallRight();
113.            }
114.        }
115.
116.
117.    }
118. } //main()
119.
120. void MoveBallRight(void) {
121.     int led_order;
122.     int run;
123.
124.     //keep the score add once

```

```

125.     int getOut = 1;
126.
127.     for (led_order = LED_PATTERNS_ORDER[5]; led_order >= LED_PATTERNS_ORDER
        [0];) {
128.
129.         run = RUNKEEP;
130.         while (run) {
131.             //check the switches changed
132.             dip_check = XGpio_DiscreteRead(&dip, 1);
133.             if (dip_check != dip_check_prev) {
134.                 xil_printf("DIP Switch Status %x, %x\r\n", dip_check_prev,
135.
                            dip_check);
136.                 dip_check_prev = dip_check;
137.                 // load timer with the new switch settings
138.                 XScuTimer_LoadTimer(TimerInstancePtr, ONE_TENTH*dip_check);
139.             }
140.
141.             //check the LEFTPADDLE pushed
142.             psb_check = XGpio_DiscreteRead(&push, 1);
143.             if (psb_check == LEFTPADDLE) {
144.                 //set StartDirection
145.                 if (led_order == LED_PATTERNS_ORDER[1]) {
146.                     StartDirection = RIGHT;
147.                     run = RUNFOBBIDEN;
148.                     led_order = LED_PATTERNS_ORDER_LEFT_OUT;
149.                 } else {
150.                     GameOver = STOP;
151.                     LED_IP_mWriteReg(XPAR_LED_IP_0_S00_AXI_BASEADDR, 0,
152.
                                LED_PATTERNS[0]);
153.                     StartDirection = RIGHT;
154.                     scoreleft += 1;
155.                     run = RUNFOBBIDEN;
156.                     led_order = LED_PATTERNS_ORDER_LEFT_OUT;
157.                     LED_IP_mWriteReg(XPAR_LED_IP_0_S00_AXI_BASEADDR, 0,
158.
                                LED_PATTERNS[led_order]);
159.                     xil_printf("Score Left = %d   Score Right = %d\r\n", sc
                        oreright, scoreleft);
160.                 }
161.             } else {
162.                 //set GameOver; display scores
163.                 if (led_order == LED_PATTERNS_ORDER[0] && getOut) {
164.                     scoreleft += 1;

```

```

165.            GameOver = STOP;
166.             StartDirection = RIGHT;
167.
168.             xil_printf("Score Left = %d   Score Right = %d\r\n",
169.                 scoreright, scoreleft);
170.             run = RUNFOBBIDEN;
171.             getOut = 0;
172.         }
173.
174.         if (XScuTimer_IsExpired(TimerInstancePtr)) {
175.             // clear status bit
176.             XScuTimer_ClearInterruptStatus(TimerInstancePtr);
177.             led_order--;
178.             LED_IP_mWriteReg(XPAR_LED_IP_0_S00_AXI_BASEADDR, 0,
179.                 LED_PATTERNS[led_order]);
180.             run = RUNFOBBIDEN;
181.         }
182.     }
183. }
184.
185. }
186. }
187.
188. void MoveBallLeft(void) {
189.     int led_order;
190.     int run;
191.
192.     //keep the score add once
193.     int getOut = 1;
194.     for (led_order = LED_PATTERNS_ORDER[1]; led_order <= LED_PATTERNS_ORDER
        [5];) {
195.         run = RUNKEEP;
196.         while (run) {
197.             dip_check = XGpio_DiscreteRead(&dip, 1);
198.             if (dip_check != dip_check_prev) {
199.                 xil_printf("DIP Switch Status %x, %x\r\n", dip_check_prev,
200.
201.                     dip_check);
202.                 dip_check_prev = dip_check;
203.                 // load timer with the new switch settings
204.                 XScuTimer_LoadTimer(TimerInstancePtr, ONE_TENTH*dip_check);
205.
206.             }
207.             psb_check = XGpio_DiscreteRead(&push, 1);

```

```

206.         if (psb_check == RIGHTPADDLE) {
207.             //set StartDirection
208.             if (led_order == LED_PATTERNS_ORDER[4]) {
209.                 StartDirection = LEFT;
210.                 run = 0;
211.                 led_order = LED_PATTERNS_ORDER_RIGHT_OUT;
212.             } else {
213.                 GameOver = STOP;
214.                 StartDirection = LEFT;
215.                 scoreright += 1;
216.                 run = RUNFOBBIDEN;
217.
218.                 led_order = LED_PATTERNS_ORDER_RIGHT_OUT;
219.                 LED_IP_mWriteReg(XPAR_LED_IP_0_S00_AXI_BASEADDR, 0,
220.                                 LED_PATTERNS[led_order]);
221.
222.                 xil_printf("Score Left = %d   Score Right = %d\r\n", sc
                    oreright, scoreleft);
223.             }
224.
225.         } else {
226.             if (led_order == LED_PATTERNS_ORDER[5] && getOut) {
227.                 //set GameOver; display scores
228.                 scoreright += 1;
229.                 GameOver = STOP;
230.                 StartDirection = LEFT;
231.                 xil_printf("Score Left = %d   Score Right = %d\r\n",
232.                             scoreright, scoreleft);
233.                 run = RUNFOBBIDEN;
234.                 getOut = 0;
235.             }
236.
237.             if (XScuTimer_IsExpired(TimerInstancePtr)) {
238.                 // clear status bit
239.                 XScuTimer_ClearInterruptStatus(TimerInstancePtr);
240.                 led_order++;
241.                 LED_IP_mWriteReg(XPAR_LED_IP_0_S00_AXI_BASEADDR, 0,
242.                                 LED_PATTERNS[led_order]);
243.                 run = RUNFOBBIDEN;
244.             }
245.         }
246.     }
247. }
248. }

```