

**Lab 4: ROM-based GameAccess Control on FPGA**

ECE 5440

**Damon Spencer**  
UH ID: 1615097

UNIVERSITYof **HOUSTON** | ECE

## Contents

<b>1 Introduction</b>	<b>3</b>
<b>2 System architecture design</b>	<b>3</b>
2.1 Top level system architecture . . . . .	3
2.2 Module symbols . . . . .	4
2.2.1 gameController module symbol (logout bonus feature included) . . . . .	4
2.2.2 authentication module symbol (logout bonus feature included) . . . . .	5
2.2.3 ram module symbol (part of password reset bonus feature) . . . . .	6
2.2.4 rom module symbol . . . . .	6
2.2.5 onemslfsr module symbol . . . . .	7
2.2.6 Score counter module symbol (bonus feature from Lab 3) . . . . .	7
2.2.7 Two digit timer module symbol . . . . .	8
2.2.8 Digit timer module symbol . . . . .	8
2.2.9 One second timer module symbol . . . . .	8
2.2.10 One hundred millisecond timer module symbol . . . . .	9
2.2.11 One millisecond timer module symbol . . . . .	9
2.2.12 Count to ten module symbol . . . . .	10
2.2.13 Count to one hundred module symbol . . . . .	10
2.2.14 Random number generator module symbol (LFSR bonus feature included) . . . . .	10
2.2.15 buttonShaper module symbol . . . . .	11
2.2.16 LoadRegister module symbol . . . . .	12
2.2.17 Seven segment decoder module symbol . . . . .	12
2.2.18 Adder module symbol . . . . .	13
<b>3 Simulation screen captures</b>	<b>13</b>
3.1 rom delay simulation screen capture . . . . .	13
3.2 authentication module with rom inside simulation screen capture . . . . .	13
3.3 one millisecond LFSR find last number simulation screen capture . . . . .	14
3.4 one millisecond LFSR comparison simulation screen capture . . . . .	14
3.5 Digit timer module simulation screen capture . . . . .	14
<b>4 oneSecondTimer module simulation screen capture</b>	<b>15</b>
<b>5 Random number generator module simulation screen capture</b>	<b>16</b>
5.1 Load register module simulation screen capture . . . . .	16
5.2 Button shaper module simulation screen capture . . . . .	16
5.3 Seven segment decoder module simulation screen capture . . . . .	16
5.4 Adder module simulation screen capture . . . . .	18
<b>6 FPGA board testing results</b>	<b>18</b>
<b>7 FPGA demonstration video</b>	<b>19</b>
7.1 Login feature demonstration . . . . .	19
7.2 Basic gameplay demonstration . . . . .	19
7.3 Password reset, logout, and login demonstration (bonus features) . . . . .	19
<b>8 Conclusion (all bonus features included for lab 4)</b>	<b>19</b>
<b>9 Appendix</b>	<b>20</b>
9.1 Verilog module code screen captures . . . . .	20
9.1.1 LFSR Random number generator module Verilog code (bonus) . . . . .	20
9.1.2 Game controller Verilog code (bonus included) . . . . .	20
9.1.3 authentication module Verilog code (bonus included) . . . . .	20

9.1.4	ram module Verilog code (part of bonus) . . . . .	20
9.1.5	rom module Verilog code . . . . .	20
9.1.6	rom module test bench Verilog code . . . . .	20
9.1.7	rom and authentication module integration test bench Verilog code . . . . .	20
9.1.8	1 millisecond LFSR timer module Verilog code . . . . .	20
9.1.9	1 millisecond LFSR timer module test bench Verilog code . . . . .	20
9.1.10	Score counter module Verilog code . . . . .	20
9.1.11	Random number generator module test bench Verilog code . . . . .	20
9.1.12	Digit timer module Verilog code . . . . .	21
9.1.13	Digit timer module testbench Verilog code . . . . .	21
9.1.14	Two Digit timer module Verilog code . . . . .	21
9.1.15	Two Digit timer module testbench Verilog code . . . . .	21
9.1.16	1 second timer module Verilog code . . . . .	21
9.1.17	1 second timer module testbench Verilog code . . . . .	21
9.1.18	100 milliseconds timer module Verilog code . . . . .	21
9.1.19	Count to 10 timer module Verilog code . . . . .	21
9.1.20	Count to 100 timer module Verilog code . . . . .	21
9.1.21	Load register module Verilog code screen capture . . . . .	21
9.1.22	Load register test bench module Verilog code screen capture . . . . .	21
9.1.23	Button shaper module Verilog code screen capture . . . . .	22
9.1.24	Button shaper test bench module Verilog code screen capture . . . . .	22
9.1.25	Seven segment decoder module Verilog code screen capture . . . . .	22
9.1.26	Seven segment decoder test bench module Verilog code screen capture . . . . .	22
9.1.27	Adder module Verilog code screen capture . . . . .	22
9.1.28	Adder test bench module Verilog code screen capture . . . . .	22

## 1 Introduction

For this lab assignment, I created an FPGA-based mental binary math game with game access control. First, the player needs to enter the last four digits of my student ID (5097). The game will not accept inputs until then. The logged out LED will also be lit at this time. To enter the password, enter the digits of my student ID one by one using the password switches. Press the password button after each digit is completed. After entering in all four digits, the logged in LED will light up and the logged out LED will switch off. To play the game, player one presses the random number generator button which will generate a random number. After doing this, the player needs to hit the player load button to load in their number. Next, the player tries to guess the binary number that when added to random number generator's number, equals F in hexadecimal. The player inputs this number using the player slide switches and then presses the player load button. The numbers the random number generator inputted, the player inputted, and the sum of the random number and the player's number numbers is outputted. If the sum of the two players numbers equals F in hexadecimal, the matching LED lights up. Otherwise, the non-matching LED lights up. If the LEDs light up, the player scores a point. The score is recorded manually. At the end of the game, the player with the highest score wins. The player can logout by pressing the player 1 load button after a round ends. The player can also reset their password by pressing the random number generator button after a round ends. To reset their password, press the random number generation button. Then flip the password switches to enter the first digit of the new password in binary. To save the first digit, hit the password button. After this, the second digit of the new password may be set. To enter the second digit, flip the password switches again with the binary version of the digit and hit the password button.

Figure 1 below shows an image of the FPGA board along with the switches and displays used:

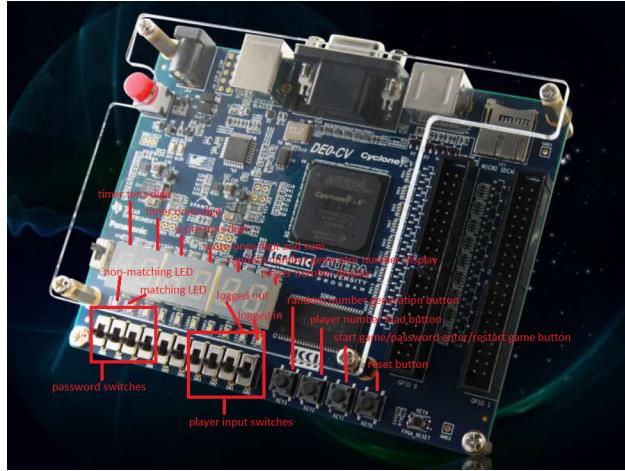


Figure 1: FPGA board picture

## 2 System architecture design

### 2.1 Top level system architecture

The top level design of my FPGA based mental binary math game with game access control uses one adder instance along with five seven segment display module instances, two buttonShaper instances, one LoadRegister instance, one rng instance, one score counter instance, one two digit timer instance, an authentication module instance, and a gameController instance. The two buttonShaper instances are connected to the player load button, and the password button respectively. The buttonShaper instances also have clock and reset inputs. The game controller instance takes in a clock input, a reset input, the output of the player button shaper instance, a timeout signal from the two digit timer, a signal that is used for the password check and password reset, and a rng button input signal. It outputs two control signals for the loadRegister

instance and the rng instance along with a signal to log out of the game, timeout enable and timer reconfig signals, and a game is over signal. The authentication module takes in a four bit password digit signal, a signal to enter in a password digit, a clock signal, a reset signal, a rom output signal, a signal that reads data from the ram, a game log out signal, and a password reset signal. The authentication module outputs a logged out signal, a logged in signal, an authentication passed signal, a rom address signal, a ram write enable signal, and a ram data write signal. The LoadRegister instance is connected directly to player one's input. The loadRegister instance also has clock and reset inputs. The output of the load register is connected to the adder instance. For the seven segment decoder, one is connected to the output of the player 1 LoadRegister, one is connected to the output of the player 2 LoadRegister, one is connected to the sum output of the adder module. Each of the player's inputs are received via four input switches. The outputs from the seven segment display module instances are displayed on the seven segment displays in the FPGA diagram above, and the adder's wongame and didnotwingame outputs are displayed on LED\_R9 and LED\_R0 respectively. The two digit timer module takes the clock, reset, timeout enable, and timer reconfig signals as inputs, and outputs the ones digit of the timer, the tens digit, and a timeout signal. The rng module takes in an rng generator signal, a clock signal, and a reset signal. It outputs a random number between 0x0 and 0xF. The rom module takes in an address and clock and outputs the data at that address to q. The ram module takes in an address, clock, input data, and a write enable switch and outputs the data at the inputted address to its q signal. To further explain this, Figure 2 below shows a diagram of the system architecture:

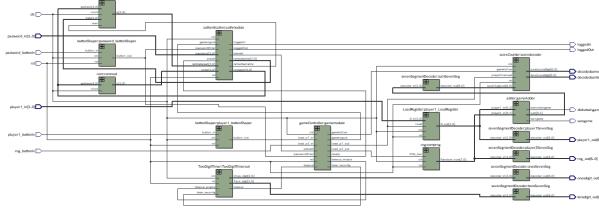


Figure 2: System architecture

## 2.2 Module symbols

### 2.2.1 gameController module symbol (logout bonus feature included)

Figure 3 below shows an image of the gameController module:

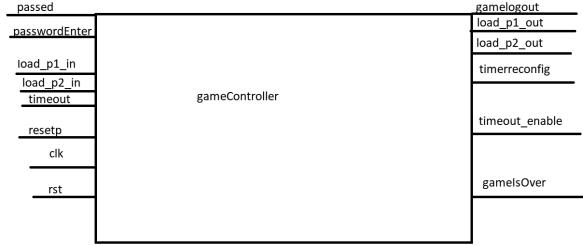


Figure 3: gameController module symbol

The gameController module uses passed, load\_p1\_in, load\_p2\_in, timeout, passwordEnter, resetp, clk, rst as inputs where passed is a signal that is one if the user passed the password check and zero otherwise, load\_p1\_in is the player load button signal, load\_p2\_in is the random number generator load signal, timeout is the input timeout signal from the timer, clk is the clock input, passwordEnter is the password entering button signal, resetp is the reset password signal, and rst is the reset input. The gameController module outputs a gameLogout signal, load\_p1\_out signal, load\_p2\_out signal, timerconfig signal, timeout\_enable signal, and gameIsOver signal where gameLogout sends a pulse if the user wants to log out of the game, load\_p1\_out is

the output load signal for the player, load\_p2\_out is the output load signal for the random number generator, timerconfig configures the timer, timeout\_enable enables the timer, and gameIsOver sends a one when the game is over and zero otherwise.

Figure 4 below shows a state machine diagram of the gameController module to help further clarify its function: Figure 5 below shows the conditions for changing state for the above state machine:

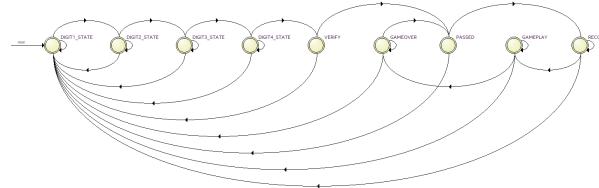


Figure 4: Game controller module state machine diagram

Condition	
1	load_p2_out=1
2	gameover
3	romq
4	passwordEnter
5	gameplay
6	reset
7	timeout_enable
8	rst
9	romq
10	passwordEnter
11	gameover
12	romq
13	passwordEnter
14	gameplay
15	reset
16	timeout_enable
17	rst
18	romq
19	passwordEnter
20	gameover
21	romq
22	passwordEnter
23	gameover
24	romq
25	passwordEnter
26	gameplay
27	reset
28	timeout_enable
29	rst
30	romq
31	passwordEnter
32	gameover
33	romq
34	passwordEnter
35	gameplay
36	reset
37	timeout_enable
38	rst
39	romq
40	passwordEnter
41	gameover
42	romq
43	passwordEnter
44	gameplay
45	reset
46	timeout_enable
47	rst
48	romq
49	passwordEnter
50	gameover
51	romq
52	passwordEnter
53	gameplay
54	reset
55	timeout_enable
56	rst
57	romq
58	passwordEnter
59	gameover
60	romq
61	passwordEnter
62	gameplay
63	reset
64	timeout_enable
65	rst
66	romq
67	passwordEnter
68	gameover
69	romq
70	passwordEnter
71	gameplay
72	reset
73	timeout_enable
74	rst
75	romq
76	passwordEnter
77	gameover
78	romq
79	passwordEnter
80	gameplay
81	reset
82	timeout_enable
83	rst
84	romq
85	passwordEnter
86	gameover
87	romq
88	passwordEnter
89	gameplay
90	reset
91	timeout_enable
92	rst
93	romq
94	passwordEnter
95	gameover
96	romq
97	passwordEnter
98	gameplay
99	reset
100	timeout_enable
101	rst
102	romq
103	passwordEnter
104	gameover
105	romq
106	passwordEnter
107	gameplay
108	reset
109	timeout_enable
110	rst
111	romq
112	passwordEnter
113	gameover
114	romq
115	passwordEnter
116	gameplay
117	reset
118	timeout_enable
119	rst
120	romq
121	passwordEnter
122	gameover
123	romq
124	passwordEnter
125	gameplay
126	reset
127	timeout_enable
128	rst
129	romq
130	passwordEnter
131	gameover
132	romq
133	passwordEnter
134	gameplay
135	reset
136	timeout_enable
137	rst
138	romq
139	passwordEnter
140	gameover
141	romq
142	passwordEnter
143	gameplay
144	reset
145	timeout_enable
146	rst
147	romq
148	passwordEnter
149	gameover
150	romq
151	passwordEnter
152	gameplay
153	reset
154	timeout_enable
155	rst
156	romq
157	passwordEnter
158	gameover
159	romq
160	passwordEnter
161	gameplay
162	reset
163	timeout_enable
164	rst
165	romq
166	passwordEnter
167	gameover
168	romq
169	passwordEnter
170	gameplay
171	reset
172	timeout_enable
173	rst
174	romq
175	passwordEnter
176	gameover
177	romq
178	passwordEnter
179	gameplay
180	reset
181	timeout_enable
182	rst
183	romq
184	passwordEnter
185	gameover
186	romq
187	passwordEnter
188	gameplay
189	reset
190	timeout_enable
191	rst
192	romq
193	passwordEnter
194	gameover
195	romq
196	passwordEnter
197	gameplay
198	reset
199	timeout_enable
200	rst
201	romq
202	passwordEnter
203	gameover
204	romq
205	passwordEnter
206	gameplay
207	reset
208	timeout_enable
209	rst
210	romq
211	passwordEnter
212	gameover
213	romq
214	passwordEnter
215	gameplay
216	reset
217	timeout_enable
218	rst
219	romq
220	passwordEnter
221	gameover
222	romq
223	passwordEnter
224	gameplay
225	reset
226	timeout_enable
227	rst
228	romq
229	passwordEnter
230	gameover
231	romq
232	passwordEnter
233	gameplay
234	reset
235	timeout_enable
236	rst
237	romq
238	passwordEnter
239	gameover
240	romq
241	passwordEnter
242	gameplay
243	reset
244	timeout_enable
245	rst
246	romq
247	passwordEnter
248	gameover
249	romq
250	passwordEnter
251	gameplay
252	reset
253	timeout_enable
254	rst
255	romq
256	passwordEnter
257	gameover
258	romq
259	passwordEnter
260	gameplay
261	reset
262	timeout_enable
263	rst
264	romq
265	passwordEnter
266	gameover
267	romq
268	passwordEnter
269	gameplay
270	reset
271	timeout_enable
272	rst
273	romq
274	passwordEnter
275	gameover
276	romq
277	passwordEnter
278	gameplay
279	reset
280	timeout_enable
281	rst
282	romq
283	passwordEnter
284	gameover
285	romq
286	passwordEnter
287	gameplay
288	reset
289	timeout_enable
290	rst
291	romq
292	passwordEnter
293	gameover
294	romq
295	passwordEnter
296	gameplay
297	reset
298	timeout_enable
299	rst
300	romq
301	passwordEnter
302	gameover
303	romq
304	passwordEnter
305	gameplay
306	reset
307	timeout_enable
308	rst
309	romq
310	passwordEnter
311	gameover
312	romq
313	passwordEnter
314	gameplay
315	reset
316	timeout_enable
317	rst
318	romq
319	passwordEnter
320	gameover
321	romq
322	passwordEnter
323	gameplay
324	reset
325	timeout_enable
326	rst
327	romq
328	passwordEnter
329	gameover
330	romq
331	passwordEnter
332	gameplay
333	reset
334	timeout_enable
335	rst
336	romq
337	passwordEnter
338	gameover
339	romq
340	passwordEnter
341	gameplay
342	reset
343	timeout_enable
344	rst
345	romq
346	passwordEnter
347	gameover
348	romq
349	passwordEnter
350	gameplay
351	reset
352	timeout_enable
353	rst
354	romq
355	passwordEnter
356	gameover
357	romq
358	passwordEnter
359	gameplay
360	reset
361	timeout_enable
362	rst
363	romq
364	passwordEnter
365	gameover
366	romq
367	passwordEnter
368	gameplay
369	reset
370	timeout_enable
371	rst
372	romq
373	passwordEnter
374	gameover
375	romq
376	passwordEnter
377	gameplay
378	reset
379	timeout_enable
380	rst
381	romq
382	passwordEnter
383	gameover
384	romq
385	passwordEnter
386	gameplay
387	reset
388	timeout_enable
389	rst
390	romq
391	passwordEnter
392	gameover
393	romq
394	passwordEnter
395	gameplay
396	reset
397	timeout_enable
398	rst
399	romq
400	passwordEnter
401	gameover
402	romq
403	passwordEnter
404	gameplay
405	reset
406	timeout_enable
407	rst
408	romq
409	passwordEnter
410	gameover
411	romq
412	passwordEnter
413	gameplay
414	reset
415	timeout_enable
416	rst
417	romq
418	passwordEnter
419	gameover
420	romq
421	passwordEnter
422	gameplay
423	reset
424	timeout_enable
425	rst
426	romq
427	passwordEnter
428	gameover
429	romq
430	passwordEnter
431	gameplay
432	reset
433	timeout_enable
434	rst
435	romq
436	passwordEnter
437	gameover
438	romq
439	passwordEnter
440	gameplay
441	reset
442	timeout_enable
443	rst
444	romq
445	passwordEnter
446	gameover
447	romq
448	passwordEnter
449	gameplay
450	reset
451	timeout_enable
452	rst
453	romq
454	passwordEnter
455	gameover
456	romq
457	passwordEnter
458	gameplay
459	reset
460	timeout_enable
461	rst
462	romq
463	passwordEnter
464	gameover
465	romq
466	passwordEnter
467	gameplay
468	reset
469	timeout_enable
470	rst
471	romq
472	passwordEnter
473	gameover
474	romq
475	passwordEnter
476	gameplay
477	reset
478	timeout_enable
479	rst
480	romq
481	passwordEnter
482	gameover
483	romq
484	passwordEnter
485	gameplay
486	reset
487	timeout_enable
488	rst
489	romq
490	passwordEnter
491	gameover
492	romq
493	passwordEnter
494	gameplay
495	reset
496	timeout_enable
497	rst
498	romq
499	passwordEnter
500	gameover
501	romq
502	passwordEnter
503	gameplay
504	reset
505	timeout_enable
506	rst
507	romq
508	passwordEnter
509	gameover
510	romq
511	passwordEnter
512	gameplay
513	reset
514	timeout_enable
515	rst
516	romq
517	passwordEnter
518	gameover
519	romq
520	passwordEnter
521	gameplay
522	reset
523	timeout_enable
524	rst
525	romq
526	passwordEnter
527	gameover
528	romq
529	passwordEnter
530	gameplay
531	reset
532	timeout_enable
533	rst
534	romq
535	passwordEnter
536	gameover
537	romq
538	passwordEnter
539	gameplay
540	reset
541	timeout_enable
542	rst
543	romq
544	passwordEnter
545	gameover
546	romq
547	passwordEnter
548	gameplay
549	reset
550	timeout_enable
551	rst
552	romq
553	passwordEnter
554	gameover
555	romq
556	passwordEnter
557	gameplay
558	reset
559	timeout_enable
560	rst
561	romq
562	passwordEnter
563	gameover
564	romq
565	passwordEnter
566	gameplay
567	reset
568	timeout_enable
569	rst
570	romq
571	passwordEnter
572	gameover
573	romq

is logged in, the passed signal outputs whether the user passed the password check or not, the romaddr outputs the selected address of the rom, the ramdatawrite signal outputs the data to write to the ram, and the ramwriteenable outputs if writing is enabled for the ram or not.

Figure 7 below shows a state machine diagram of the authentication module to help further clarify its function: Figure 8 below shows the conditions for changing state for the above state machine:

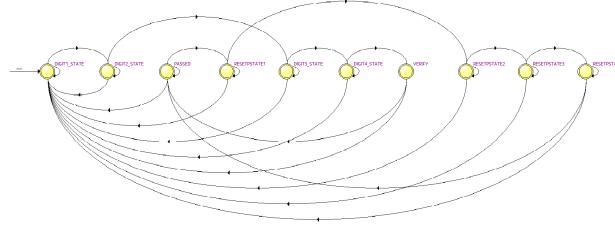


Figure 7: Authentication module state machine diagram

Source State	Destination State	Condition	Source State	Destination State	Condition
1 DIGIT1_STATE	DIGIT2_STATE	[count[0]>=0 & count[1]>=0 & count[2]>=0 & count[3]>=0]	7 DIGIT1_STATE	DIGIT2_STATE	[count[0]>=0 & count[1]>=0 & count[2]>=0 & count[3]>=0]
2 DIGIT1_STATE	DIGIT2_STATE	[count[0]>=0 & count[1]>=0 & count[2]>=0 & count[3]>=0]	8 DIGIT1_STATE	DIGIT2_STATE	[count[0]>=0 & count[1]>=0 & count[2]>=0 & count[3]>=0]
3 DIGIT1_STATE	DIGIT2_STATE	[count[0]>=0 & count[1]>=0 & count[2]>=0 & count[3]>=0]	9 DIGIT1_STATE	VERIFY	[count[0]>=0 & count[1]>=0 & count[2]>=0 & count[3]>=0]
4 DIGIT1_STATE	DIGIT3_STATE	[count[0]>=0 & count[1]>=0 & count[2]>=0 & count[3]>=0]	10 DIGIT1_STATE	DIGIT2_STATE	[hv0]
5 DIGIT1_STATE	DIGIT3_STATE	[count[0]>=0 & count[1]>=0 & count[2]>=0 & count[3]>=0]	11 DIGIT1_STATE	DIGIT2_STATE	[count[0]>=0 & count[1]>=0 & count[2]>=0 & count[3]>=0]
6 DIGIT1_STATE	DIGIT3_STATE	[count[0]>=0 & count[1]>=0 & count[2]>=0 & count[3]>=0]	12 PASSED	DIGIT1_STATE	[preset]&[gmeigou]&[hv0]
7 DIGIT1_STATE	DIGIT3_STATE	[count[0]>=0 & count[1]>=0 & count[2]>=0 & count[3]>=0]	13 PASSED	RESETSTATE1	[preset]&[gmeigou]&[hv0]
8 DIGIT1_STATE	DIGIT3_STATE	[count[0]>=0 & count[1]>=0 & count[2]>=0 & count[3]>=0]	14 PASSED	PASSED	[preset]&[gmeigou]&[hv0]
9 DIGIT1_STATE	DIGIT3_STATE	[count[0]>=0 & count[1]>=0 & count[2]>=0 & count[3]>=0]	15 RESETSTATE1	DIGIT1_STATE	[hv0]
10 DIGIT1_STATE	DIGIT3_STATE	[count[0]>=0 & count[1]>=0 & count[2]>=0 & count[3]>=0]	16 RESETSTATE1	RESETSTATE2	[hv0]
11 DIGIT1_STATE	DIGIT3_STATE	[count[0]>=0 & count[1]>=0 & count[2]>=0 & count[3]>=0]	17 RESETSTATE1	RESETSTATE2	[count[0]>=0 & count[1]>=0 & count[2]>=0 & count[3]>=0]
12 PASSED	DIGIT1_STATE	[pmeigou]&[hv0]&[gmeigou]	18 RESETSTATE2	DIGIT1_STATE	[count[0]>=0 & count[1]>=0 & count[2]>=0 & count[3]>=0]
13 PASSED	RESETSTATE1	[pmeigou]&[hv0]&[gmeigou]	19 RESETSTATE2	DIGIT1_STATE	[hv0]
14 PASSED	PASSED	[pmeigou]&[hv0]&[gmeigou]	20 RESETSTATE2	RESETSTATE2	[count[0]>=0 & count[1]>=0 & count[2]>=0 & count[3]>=0]
15 RESETSTATE1	DIGIT1_STATE	[hv0]	21 RESETSTATE2	RESETSTATE2	[count[0]>=0 & count[1]>=0 & count[2]>=0 & count[3]>=0]
16 RESETSTATE1	RESETSTATE2	[hv0]	22 RESETSTATE2	RESETSTATE2	[count[0]>=0 & count[1]>=0 & count[2]>=0 & count[3]>=0]
17 RESETSTATE1	RESETSTATE2	[hv0]	23 RESETSTATE2	DIGIT1_STATE	[hv0]
18 RESETSTATE1	RESETSTATE2	[hv0]	24 RESETSTATE4	RESETSTATE4A	[count[0]>=0 & count[1]>=0 & count[2]>=0 & count[3]>=0]
19 RESETSTATE1	RESETSTATE2	[hv0]	25 RESETSTATE4	DIGIT1_STATE	[hv0]
20 RESETSTATE2	RESETSTATE2	[hv0]	26 RESETSTATE4A	PASSED	[count[0]>=0 & count[1]>=0 & count[2]>=0 & count[3]>=0]
21 RESETSTATE2	RESETSTATE2	[hv0]	27 VERIFY	DIGIT1_STATE	[Equal1]&[Equal2]&[Equal3]&[Equal4]&[Equal5]&[Equal6]&[Equal7]&[Equal8]&[Equal9]
22 RESETSTATE2	RESETSTATE2	[hv0]	28 VERIFY	PASSED	[Equal1]&[Equal2]&[Equal3]&[Equal4]&[Equal5]&[Equal6]&[Equal7]&[Equal8]&[Equal9]
23 RESETSTATE2	RESETSTATE2	[hv0]			
24 RESETSTATE4	RESETSTATE4A	[hv0]			
25 RESETSTATE4	DIGIT1_STATE	[hv0]			
26 RESETSTATE4A	PASSED	[hv0]			
27 VERIFY	DIGIT1_STATE	[Equal1]&[Equal2]&[Equal3]&[Equal4]&[Equal5]&[Equal6]&[Equal7]&[Equal8]&[Equal9]			
28 VERIFY	PASSED	[Equal1]&[Equal2]&[Equal3]&[Equal4]&[Equal5]&[Equal6]&[Equal7]&[Equal8]&[Equal9]			

Figure 8: Authentication module state machine transition table

### 2.2.3 ram module symbol (part of password reset bonus feature)

Figure 9 below shows an image of the ram module:

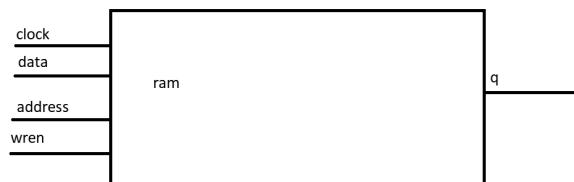


Figure 9: ram module symbol

The ram module uses clk, address, data, and wren as inputs where clk is the clock input address is the address of the rom word that the program wants to read, data is the data the program wants to write to the ram, and write enable enables writing to the ram if it is 1, and reading otherwise. It outputs the word at the address that is inputted to the ram module to q.

### 2.2.4 rom module symbol

Figure 10 below shows an image of the rom module:

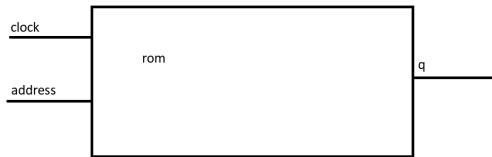


Figure 10: rom module symbol

The rom module uses clk, and address as inputs where clk is the clock input and address is the address of the rom word that the program wants to read. It outputs the word at the address that is inputted to the rom module to q.

### 2.2.5 onemslfsr module symbol

Figure 11 below shows an image of the onemslfsr module:



Figure 11: One millisecond LFSR timer symbol

The onemslfsr module uses clk, rst, and enable as inputs where clk is the clock input, reset is the reset input, enable is the input that turns the timer on, and onems\_Timerout is the output of the module and is a one cycle pulse every one millisecond. The timer uses an internal linear feedback shift register to operate the timer.

### 2.2.6 Score counter module symbol (bonus feature from Lab 3)

Figure 12 below shows the module symbol for my score counter, which is used for the bonus feature:

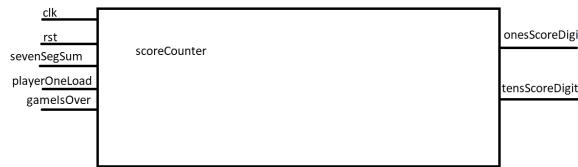


Figure 12: Score Counter module symbol

This module implements automatic scoring for the FPGA based mental binary math game. The scoreCounter module receives a clock input as clk, a reset input as rst, the sum of the players number and the random number generator's number decoded into a format suitable for input into a seven segment display as sevenSegSum, the player one button signal outputted from the access controller as playerOneLoad, and a game over signal from the access controller as gameIsOver. Note gameIsOver is one if the game is over and zero otherwise. onesScoreDigit outputs the ones digit of the player's score decoded into a format suitable for display on a seven segment display if the game is over, and the sum of the player's number and the random number generator's number otherwise. tensScoreDigit displays outputs the tens digit of the player's score decoded into a format suitable for display on a seven segment display if the game is over,

and 0b1111111 so that the seven segment display appears off otherwise. The module also uses two internal sevenSegmentDecoder modules.

### 2.2.7 Two digit timer module symbol

Figure 13 below shows the module symbol for my two digit timer:



Figure 13: Two digit timer module symbol

This module creates a two digit timer for the FPGA. It counts down from 99 seconds to zero and sends a timeout signal when time is up. The module takes a clock signal labeled clk, a reset signal labeled rst, a timer enable signal labeled timeout\_enable, and a timer reconfiguration signal called timer\_reconfig as inputs. The module outputs the ones digit of the current time remaining as Ones\_digit, the tens digit of the current time remaining as Tens\_digit, and the timeout signal that is one when time is up and zero otherwise as timeout. Internally, the timer module uses two one digit timer modules connected together which are driven by a one second timer that goes off every one second. When 99 seconds elapse, the timer sends a timeout signal.

### 2.2.8 Digit timer module symbol

Figure 14 below shows the module symbol for my two digit timer:

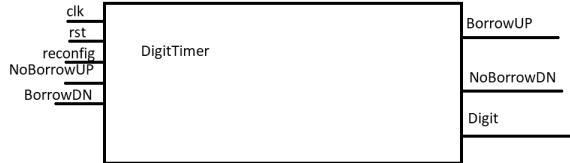


Figure 14: Digit timer module symbol

This module creates a one digit timer for the FPGA. It counts down from 9 to zero and sets NoBorrowDN to one when time runs out. The module takes a clock signal labeled clk, a reset signal labeled rst, a reconfiguration signal labeled reconfig, a NoBorrowUP signal that is one when the digit immediately greater than the current digit timer instance cannot be borrowed from and zero otherwise, and a BorrowDN signal that is a one clock cycle pulse when the digit immediately smaller than the current timer would like to borrow a one as inputs. The module outputs the BorrowUP signal when it would like to borrow a one from the digit immediately larger than the current digit, a NoBorrowDN signal when the digit timer that represents a digit immediately to the left of the current digit timer instance cannot borrow, and a Digit signal which outputs the current digit of the digit timer instance.

### 2.2.9 One second timer module symbol

Figure 15 below shows the module symbol for my one second timer:

This

This module generates a one second timer for the FPGA. The one second timer takes a clock signal called clk, a reset signal called rst, and a timer enable signal called enable as inputs. The enable signal is one when the timer is enabled, and zero otherwise. It outputs a timeout pulse when one second elapses. Internally, the one second timer module passes the enable, clock, and reset signals to an internal one hundred millisecond



Figure 15: One second timer module symbol

timer module. The output of the internal one hundred millisecond module is passed to an internal module called count to 10 that sends an output pulse when 10 input pulses have been received. When the internal count to 10 module sends out an output pulse, one second has elapsed.

### 2.2.10 One hundred millisecond timer module symbol

Figure 16 below shows the module symbol for my one hundred millisecond timer:



Figure 16: One hundred millisecond timer module symbol

This

This module generates a one hundred millisecond timer for the FPGA. The one hundred millisecond timer takes a clock signal called clk, a reset signal called rst, and a timer enable signal called enable as inputs. The enable signal is one when the timer is enabled, and zero otherwise. It outputs a timeout pulse when one second elapses. Internally, the one hundred millisecond timer module passes the enable, clock, and reset signals to an internal one millisecond timer module. The output of the internal one millisecond module is passed to an internal module called count to 100 that sends an output pulse when 100 input pulses have been received. When the internal count to 100 module sends out an output pulse, one hundred milliseconds have elapsed.

### 2.2.11 One millisecond timer module symbol

Figure 17 below shows the module symbol for my one millisecond timer:

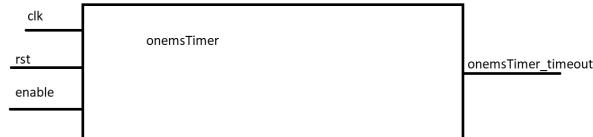


Figure 17: One millisecond timer module symbol

This

This module generates a one millisecond timer for the FPGA. The one millisecond timer takes a clock signal called clk, a reset signal called rst, and a timer enable signal called enable as inputs. The enable signal is one when the timer is enabled, and zero otherwise. It outputs a timeout pulse when one second elapses. Internally, the one millisecond timer module produces a timeout signal every 50,000 clock cycles and by then one millisecond elapses.

### 2.2.12 Count to ten module symbol

Figure 18 below shows the module symbol for my count to ten module:

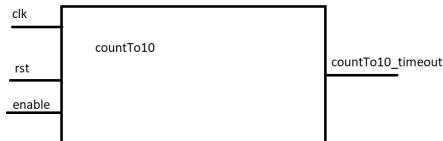


Figure 18: Count to ten module symbol

This module generates a one clock cycle pulse every time ten input pulses are received. The countTo10 module takes a clock signal called clk, a reset signal called rst, and a timer enable signal called enable as inputs. The enable signal is one when the timer is enabled, and zero otherwise.

### 2.2.13 Count to one hundred module symbol

Figure 19 below shows the module symbol for my count to one hundred module:

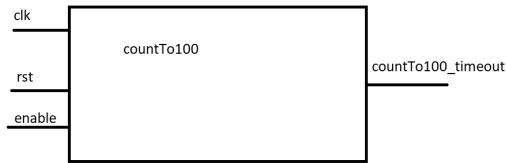


Figure 19: Count to one hundred module symbol

This module generates a one clock cycle pulse every time one hundred input pulses are received. The countTo100 module takes a clock signal called clk, a reset signal called rst, and a timer enable signal called enable as inputs. The enable signal is one when the timer is enabled, and zero otherwise.

### 2.2.14 Random number generator module symbol (LFSR bonus feature included)

Figure 20 below shows the module symbol for my random number generator which uses a LFSR:

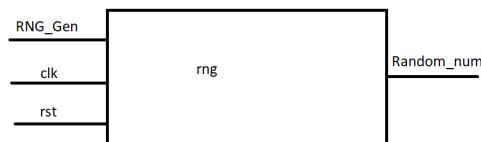


Figure 20: Digit timer module symbol

This module generates a random number for the FPGA. The random number generator module takes a signal that tells it to generate a random number, a clock signal as clk, and a reset signal as rst as inputs. The RNG\_Gen signal is zero when the random number generator should generate a random number, and one otherwise. It is a button press from a physical button. The Random\_num signal is the only output, and is a 4 bit random number. The random number generator works by incrementing a LFSR for each clock

cycle that the RNG\_Gen signal is zero. During this time the Random\_num signal will change between 0x0 and 0xF as the LFSR is changed. Random\_num is sourced from several bits of the LFSR. When Rng\_Gen signal is one, the current Random\_num value will be outputted without being changed. Since there is no way to know exactly how many clock cycles a button press will be for, the output will be a random number.

### 2.2.15 buttonShaper module symbol

Figure 21 below shows an image of the buttonShaper module:



Figure 21: Button shaper module symbol

The purpose of this module is to turn an input button press into a single clock cycle pulse. button\_in is the input signal for the button press, rst is the reset button input, clk is the FPGA clock input, and button\_out is the button pulse output. Figure 22 below shows a state machine diagram of the buttonShaper module to help further clarify it's function:

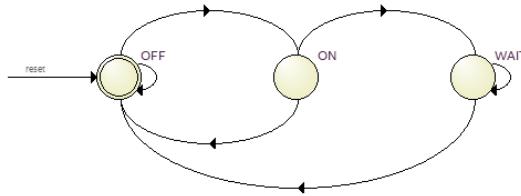


Figure 22: Button shaper module state machine diagram

Figure 23 below shows the conditions for changing state for the above state machine:

Source State	Destination State	Condition
1 OFF	OFF	$(\neg \text{button\_in}) \cdot (\neg \text{rst}) + (\text{button\_in})$
2 OFF	ON	$(\neg \text{button\_in}) \cdot (\text{rst})$
3 ON	OFF	$(\text{rst})$
4 ON	WAIT	$(\text{rst})$
5 WAIT	OFF	$(\neg \text{button\_in}) \cdot (\neg \text{rst}) + (\text{button\_in})$
6 WAIT	WAIT	$(\neg \text{button\_in}) \cdot (\text{rst})$

Figure 23: Button shaper module state machine transition table

In addition, Figure 24 below shows the expected waveform for the button shaper module:

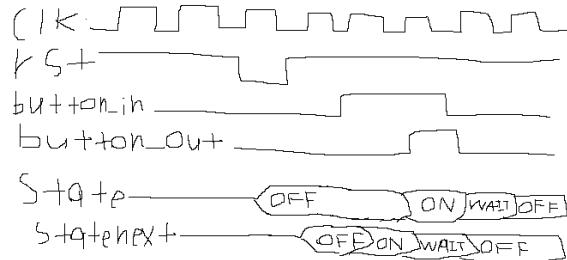


Figure 24: Button shaper module waveform

### 2.2.16 LoadRegister module symbol

Figure 25 below shows an image of the LoadRegister module:



Figure 25: Load register module symbol

The purpose of this module is to store a four bit number into a register for later use. It is controlled via the load, clk, and rst signals. D<sub>in</sub> is the number LoadRegister stores in its register, D<sub>out</sub> is the number outputted by the register, Load is the load signal, rst is the reset button input, and clk is the FPGA clock input.

### 2.2.17 Seven segment decoder module symbol

Figure 26 below shows the module symbol for my seven segment decoder:



Figure 26: Seven segment decoder module symbol

The purpose of the module is to allow a four bit input to be displayed on a seven segment display. There are two signals for the module: decoder\_in, and decoder\_out. decoder\_in is a four bit input, and decoder\_out is a seven bit output that maps the input bits to a seven segment display digit.

### 2.2.18 Adder module symbol

Figure 27 below shows the module symbol for my adder:

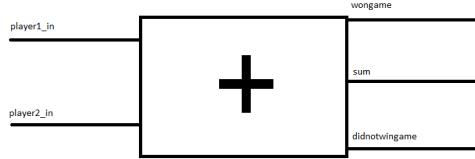


Figure 27: Adder module symbol

The purpose of the module is to allow two four bit numbers to be added together. The module receives two four bit numbers player1\_in and player2\_in as inputs. It outputs two one bit numbers wongame and didnotwingame along with one seven bit number sum. wongame is one if the sum of player1\_in and player2\_in is 0xF, and zero otherwise. didnotwingame is zero if the sum of player1\_in and player2\_in is 0xF, and one otherwise. sum is the sum of the binary numbers player1 and player2 inputted.

## 3 Simulation screen captures

### 3.1 rom delay simulation screen capture

Figure 28 below shows a screen capture of my rom module simulation that finds the propagation delay of the rom: To find the propagation delay for the rom, I change the rom read address every few clock cycles and

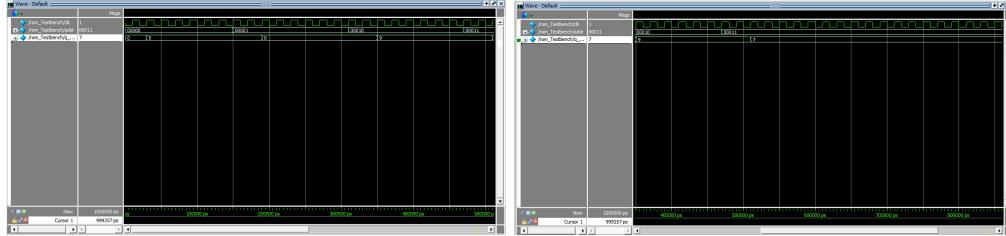


Figure 28: rom delay simulation screen capture

visually determine how many cycles later the q output of the rom changes. I determined the propagation delay to be 2 clock cycles.

### 3.2 authentication module with rom inside simulation screen capture

Figure 29 below shows a screen capture of my rom module simulation that tests the rom while connected to the authentication module: To test the rom, I tested the login process of the authentication module with the rom connected. I entered the rom password 5097. After entering each digit in binary using the password switches, I hit the password button. This was repeated four times to enter in all digits. After doing this, I was able to successfully log into the device with the rom and the rom q bus displayed every output correctly during the right time.

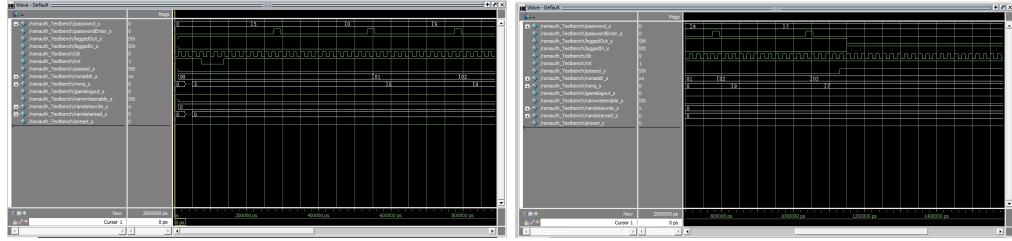


Figure 29: authentication module with rom inside simulation screen capture

### 3.3 one millisecond LFSR find last number simulation screen capture

Figure 30 below shows a screen capture of my onemlsfr module simulation that finds the last value of the lfsr used: To find the value for the lfsr 1 millisecond timer that occurs when the counter based one millisecond



Figure 30: one millisecond lfsr find last number simulation screen capture

timer has a timeout pulse, I created a testbench that runs the clock for 50,000 cycles after a reset. I then search for the clock pulse that the one millisecond timer produces when one millisecond occurs and take the value that the lfsr has at that time as the last value that the lfsr should have after one millisecond. This value ends up being 0b1101101101101100. Note that due to the way my timer is designed, I actually need to use the value of the LFSR one clock cycle before this which is 0b0110110110110110.

### 3.4 one millisecond LFSR comparison simulation screen capture

Figure 31 below shows a screen capture of my onemlsfr module simulation that compares the LFSR based timer to the counter based timer for accuracy: To test the LFSR based timer, I verified that it produced a timeout pulse at the same time as the counter based timer. The timeout pulses for both modules occurred at the same time, which is exactly what I would like.

### 3.5 Digit timer module simulation screen capture

Figure 32 below shows a screen capture of my digit timer module simulation:

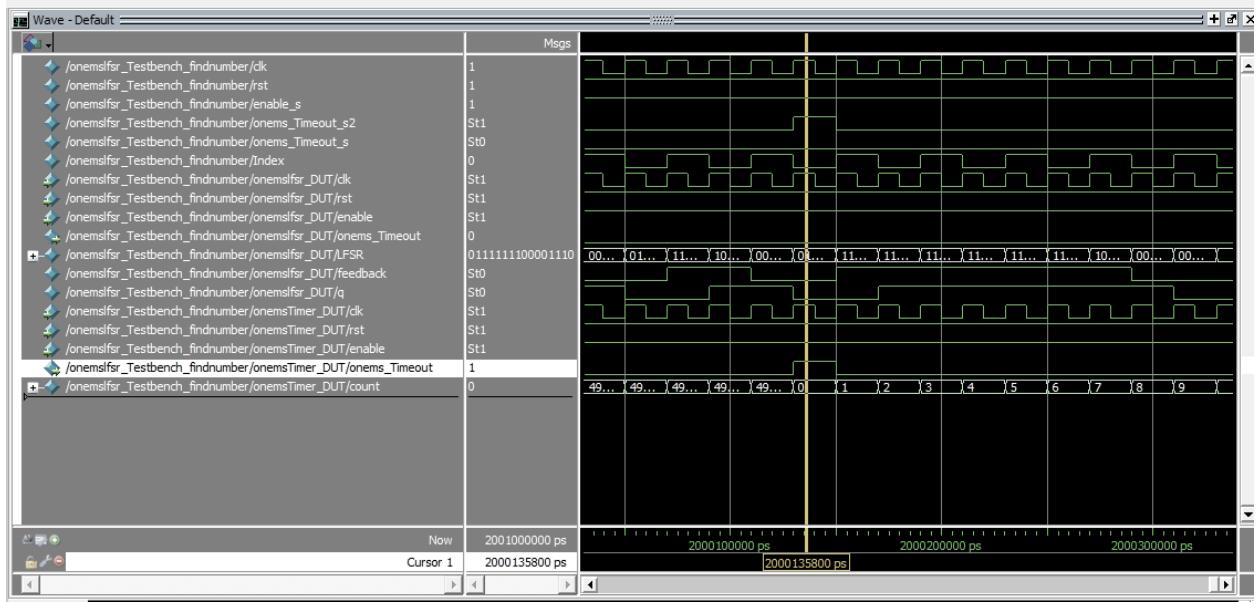


Figure 31: one millisecond lfsr comparison simulation screen capture

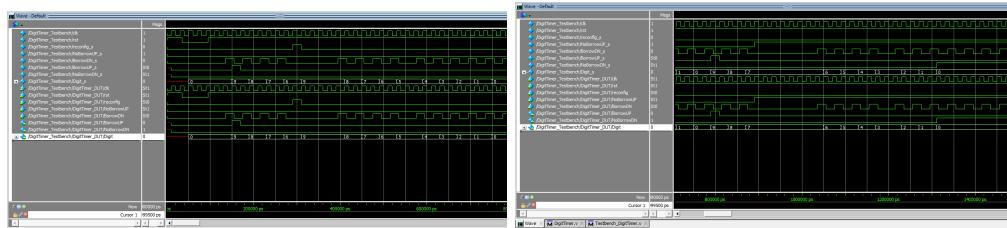


Figure 32: Digit timer module simulation screen capture

The simulation thoroughly tests the digit timer module. The digit timer module is tested with multiple borrowDn pulses and a reconfig signal pulse. The NoBorrowUP signal is also changed in the second half of the simulation from 0 to 1. Initially when running the simulation, the digit is decremented as the borrowDN signal inputs pulse signals. Next, the reconfig signal is used, and the digit is reset to 9. After this, the digit is decremented to 0 and borrows a 1 from the next largest digit. After doing this, it continues to decrement as more borrowDN pulses are inputted. The NoBorrowUP signal is then set to 1 and borrowDN pulses continue to be inputted. Eventually, the digit in the digit timer reaches zero, and the NoBorrowDN flag is set to 1. Overall, the digit timer works correctly.

## 4 oneSecondTimer module simulation screen capture

Figure 33 below shows a screen capture of my oneSecondTimer module simulation: To test the one second

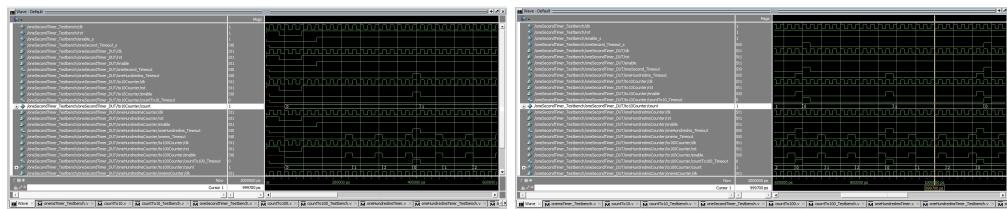


Figure 33: 1 second timer module simulation screen capture

timer, I created a test bench that temporarily lowers reset to zero, sets reset back to one, then sets enable to one for enough clock cycles to see two iterations of the one second timer. The critical case occurs when the one second timer produces a timeout signal. This causes the timeout signal to go high for one clock cycle. In this simulation, the durations of the 1msTimer, countTo10 timer, and countTo100 timer were reduced to 4, 2, and 3 clock cycles each respectively to make the modules easier to test. Since the 1 ms timer was an input to the countTo100 timer and the countTo100 timer was an input to the countTo10 timer, the 1 second timer should take 24 clock cycles to go off. In the simulation, the one second timer goes off as expected, which is great.

## 5 Random number generator module simulation screen capture

Figure 34 below shows a screen capture of my rng module simulation: To test the random number generator,

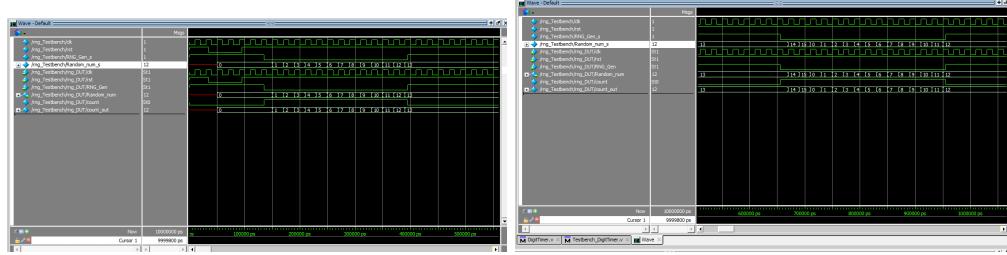


Figure 34: random number generator module simulation screen capture

I created a test bench that sets RNG\_Gen equal to zero for thirteen clock cycles, then sets RNG\_Gen to one for thirteen more cycles, sets RNG\_Gen to zero for fifteen cycles, and finally sets RNG\_Gen to one for four clock cycles. The Random number generator first increments its internal counter and its output settles on 13 when the random number generator input is one. Next the random number generator input is set low again. After finishing its second low signal, the output of the random number generator ends up at 12 during the second round. Overall, the random number generator worked correctly.

### 5.1 Load register module simulation screen capture

Figure 35 below shows a screen capture of my load register module simulation:

The simulation tests the LoadRegister module quite well. The testbench tests two different instances where a load signal is inputted, with false inputs created during the process. One of the most important tests is what happens when the first load signal is inputted. When the first load signal is inputted, D\_in = 4'b0001. One clock cycle later, D\_out becomes 4'b0001. For several cycles, D\_in is changed without a load signal and D\_out does not change. This demonstrates my module works very well.

### 5.2 Button shaper module simulation screen capture

Figure 36 below shows a screen capture of my button shaper module simulation:

To test the buttonShaper module, I first simulated a reset button press. From the above screenshot, the state variable only has an unknown value before the reset button is pressed, which is good. Next, I simulated two button presses of differing lengths. In both cases the button shaper module produced a one clock cycle pulse. Overall, I think my button shaper module works correctly.

### 5.3 Seven segment decoder module simulation screen capture

Figure 37 below shows screen captures of my seven segment decoder module simulation:

The simulation tests all 16 cases of the seven segment decoder module. Note for each case, a segment of the seven segment decoder is lit up when the bit corresponding to that segment is zero and is off when the bit corresponding to that segment is one. To test the module code, I compared the input of the module to

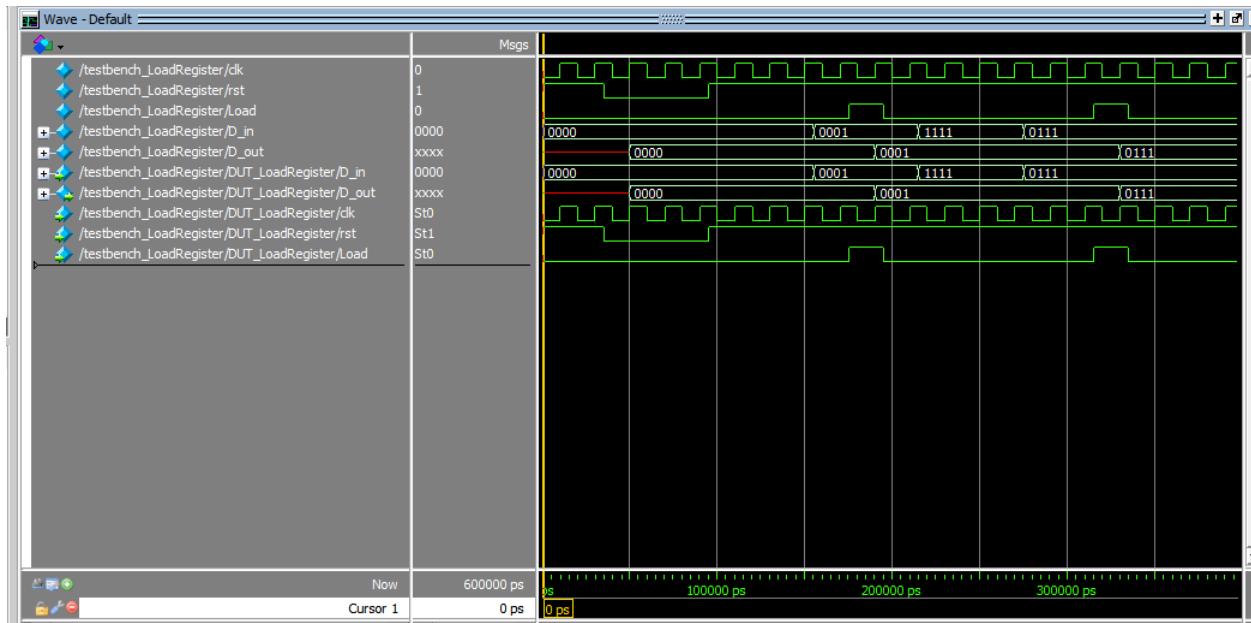


Figure 35: Load register module simulation screen capture

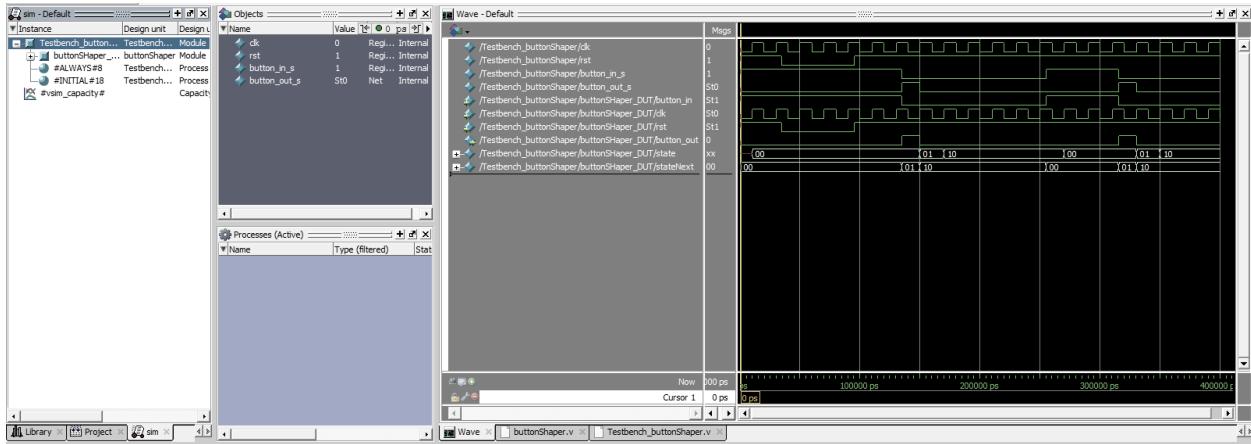


Figure 36: Button Shaper module simulation screen capture

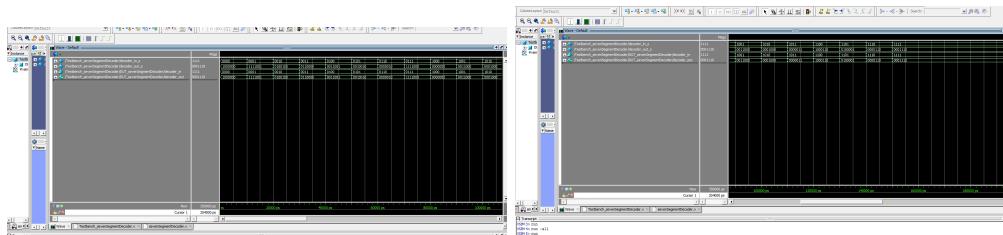


Figure 37: Seven segment decoder module simulation screen capture

the expected output of the module. Every single case outputted exactly what I expected, so I was able to believe my module code was correct. One good benchmark for the module code is inputting 0b1000 should output the binary number 0b0000000. 0b0000000 corresponds to an eight on the seven segment display. My module correctly outputs the number 0b0000000 when the input is 0b1000, so this benchmark case is correct.

## 5.4 Adder module simulation screen capture

Figure 38 below shows a screen capture of my adder module simulation:

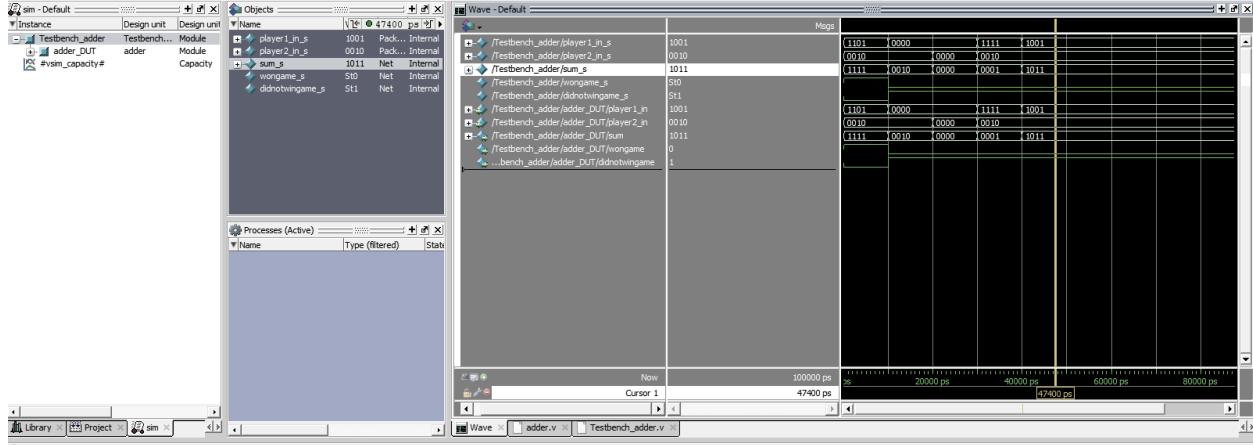


Figure 38: Adder module simulation screen capture

The simulation tests five different cases for the adder module. Below I will describe two critical test cases: a case when the sum of the player's two numbers adds up to 0xF, and another case where the player's two numbers do not add up to 0xF.

To test what happens when the player's numbers add up to 0xF, I created a test case where player 1 inputted 0b1101, and player 2 inputted 0b0010. In that case, the seven segment display with player 1's number outputted player 1's number, the seven segment display with player 2's number outputted player 2's number, and the seven segment display with the sum of both player's numbers outputted 0xF, the won game LED lit up, and the did not win game led did not light up.

To test what happens when the player's numbers do not add up to 0xF, I created a test case where player 1 inputted 0b0000, and player 2 inputted 0b0010. In that case, the seven segment display with player 1's number outputted player 1's number, the seven segment display with player 2's number outputted player 2's number, and the seven segment display with the sum of both player's numbers outputted 0x1, the won game LED did not light up, and the did not win game led lit up.

Overall, the simulation went exactly as I expected.

## 6 FPGA board testing results

Below are eight images displaying my FPGA board testing results. Figure 39 below shows an image of the FPGA right after pressing the reset button:

Figure 40 below shows an image of the FPGA after logging in with the password 5097. To log in, I entered each digit with the slide switches and then pressed the passwordEnter button after each digit. This happened four times until I was logged in. The timer is then set for 99 seconds:

Note the loggedIn LED turns on and the loggedOut LED turns off.

Next, I hit the game start button to start the game. I then hit the random number generation button to generate a random number. 0xd was generated. I then entered 0x1 and the sum display was set to 0xE. The timer had 48 seconds remaining in this photo.

Figure 41 below shows an image of the FPGA after hitting the random number generator button, entering in 0x1 for the player, and hitting the player load button:

Next, I changed the player input to 0x2 and the sum was set to 0xF. The wongame led lit up while the didnotwingame led turned off. The timer had 42 seconds remaining in this photo. Figure 42 below shows an image of the FPGA after entering in 0x2 for the player and hitting the player load button:

After changing the player input, I waited for the timer to expire. The time ran out and the timer displayed 0 seconds remaining. The score of 1 was then displayed on the FPGA score seven segment displays

where the sum was displayed. Figure 43 below demonstrates this functionality: I then played another round and allowed the timer to expire again. After that, I hit the random number generator button to change the password for the player. I changed the password to 1234. Figure 44 below demonstrates the password change functionality: Next, I proceeded to logout with the password changed. Figure 45 below demonstrates the logout functionality: Finally, I log back in with the new password 1234. The login functionality works as expected with the new password. Figure 46 below demonstrates the login functionality:

## 7 FPGA demonstration video

### 7.1 Login feature demonstration

I broke the demonstration videos of my FPGA into several sections. Below is a link to the login feature of the FPGA:

[https://drive.google.com/file/d/15xfrzqKUT-PPZI7NQez0g\\_5lVkJhNRMT/view?usp=sharing](https://drive.google.com/file/d/15xfrzqKUT-PPZI7NQez0g_5lVkJhNRMT/view?usp=sharing)

### 7.2 Basic gameplay demonstration

Below is a video demonstration of the basic gameplay and automatic scoring feature of my FPGA based mental binary math game: <https://drive.google.com/file/d/162o6GW0tHPDkY-n2398Zc2Oa6vipB22e/view?usp=sharing>

### 7.3 Password reset, logout, and login demonstration (bonus features)

Below is a video demonstration of the password reset and logout functionality along with an additional login for testing: [https://drive.google.com/file/d/164yCRNxPfxuzE\\_Fu2MQgd6oxnd\\_em7Xc/view?usp=sharing](https://drive.google.com/file/d/164yCRNxPfxuzE_Fu2MQgd6oxnd_em7Xc/view?usp=sharing)

## 8 Conclusion (all bonus features included for lab 4)

Overall my lab 4 works perfectly. There are no known issues, and all three bonus features for the lab are implemented.

## 9 Appendix

### 9.1 Verilog module code screen captures

#### 9.1.1 LFSR Random number generator module Verilog code (bonus)

Figure 47 below shows a screen capture of my rng module code with an internal LFSR: The random number generator with a LFSR works quite well.

#### 9.1.2 Game controller Verilog code (bonus incuded)

Figure 48 below shows a screen capture of my game controller module code: My game controller module works as expected.

#### 9.1.3 authentication module Verilog code (bonus included)

Figure 49 below shows a screen capture of my screen capture of my romauth\_Testbench module code: The test bench shows the new authentication module works well.

#### 9.1.4 ram module Verilog code (part of bonus)

Figure 50 below shows a screen capture of my ram module code: My ram module works as expected.

#### 9.1.5 rom module Verilog code

Figure 51 below shows a screen capture of my rom module code: My rom module works as expected.

#### 9.1.6 rom module test bench Verilog code

Figure 52 below shows a screen capture of my screen capture of my rom\_Testbench module code: The test bench does a great job at finding the propagation delay of the rom module.

#### 9.1.7 rom and authentication module integration test bench Verilog code

Figure 53 below shows a screen capture of my screen capture of my romauth\_Testbench module code: The test bench shows the rom works well with the authentication module.

#### 9.1.8 1 millisecond LFSR timer module Verilog code

Figure 54 below shows a screen capture of my onemslfsr module code: My linear feedback shift register is significantly more efficient than the counter based timer.

#### 9.1.9 1 millisecond LFSR timer module test bench Verilog code

Figure 55 below shows a screen capture of my screen capture of my onemslfsr\_testbench\_findnumber module code: The test bench does a great job at finding the last value needed in the LFSR for the 1 millisecond timer.

#### 9.1.10 Score counter module Verilog code

Figure 56 below shows a screen capture of my scoreCounter module code: The score counter efficiently counts the player's score without errors.

#### 9.1.11 Random number generator module test bench Verilog code

Figure 57 below shows a screen capture of my OneSecondTimer testbench module code: The test bench for the random number generator demonstrates the random number generator's capabilities well.

### **9.1.12 Digit timer module Verilog code**

Figure 58 below shows a screen capture of my DigitTimer module code: The digit timer accurately keeps track of a single digit number.

### **9.1.13 Digit timer module testbench Verilog code**

Figure 59 below shows a screen capture of my DigitTimer testbench module code: The digit timer testbench does a good job of testing the digit timer's capabilities.

### **9.1.14 Two Digit timer module Verilog code**

Figure 60 below shows a screen capture of my TwoDigitTimer module code: The two digit timer works well for a 99 second timer.

### **9.1.15 Two Digit timer module testbench Verilog code**

Figure 61 below shows a screen capture of my DigitTimer testbench module code: This module thoroughly tests the two digit timer module.

### **9.1.16 1 second timer module Verilog code**

Figure 62 below shows a screen capture of my oneSecondTimer module code: The one second timer module counts seconds extremely well.

### **9.1.17 1 second timer module testbench Verilog code**

Figure 63 below shows a screen capture of my OneSecondTimer testbench module code: The one second timer module testbench uses a shortened one second timer to test out the capabilities of the one second timer module.

### **9.1.18 100 milliseconds timer module Verilog code**

Figure 64 below shows a screen capture of my oneHundredmsTimer module code: The 100 milliseconds timer module goes off every 100 millisecond and works very well.

### **9.1.19 Count to 10 timer module Verilog code**

Figure 65 below shows a screen capture of my countTo10 module code: The count to 10 timer module uses a four bit counter to count to 10.

### **9.1.20 Count to 100 timer module Verilog code**

Figure 66 below shows a screen capture of my countTo100 module code: The count to 100 module accurately counts to 100.

### **9.1.21 Load register module Verilog code screen capture**

Figure 67 below shows screenshots of my load register module Verilog code:

The load register module does a good job at storing number for later use.

### **9.1.22 Load register test bench module Verilog code screen capture**

Figure 68 below shows screenshots of my load register test bench module Verilog code:

The load register module test bench tests two different cases for the load register module and both work as expected.

**9.1.23 Button shaper module Verilog code screen capture**

Figure 69 below shows screenshots of my button shaper module Verilog code:

The button shaper is quite accurate at turning an arbitrary button press into a single clock cycle pulse.

**9.1.24 Button shaper test bench module Verilog code screen capture**

Figure 70 below shows screenshots of my button shaper test bench module Verilog code:

The button shaper test bench tests multiple cases with arbitrary length button inputs. Overall, the button shaper module works well.

**9.1.25 Seven segment decoder module Verilog code screen capture**

Figure 71 below shows a screenshot of my seven segment decoder module Verilog code:

Since the input is four bits, I needed sixteen input combinations to test all cases.

**9.1.26 Seven segment decoder test bench module Verilog code screen capture**

Figure 72 below shows a screenshot of my seven segment decoder test bench module Verilog code:

The seven segment decoder module test bench includes all sixteen cases.

**9.1.27 Adder module Verilog code screen capture**

Figure 73 below shows a screenshot of my adder module Verilog code:

The adder module includes all bonus features.

**9.1.28 Adder test bench module Verilog code screen capture**

Figure 74 below shows a screenshot of my adder test bench module Verilog module code:

The test bench has five different test cases for the adder.

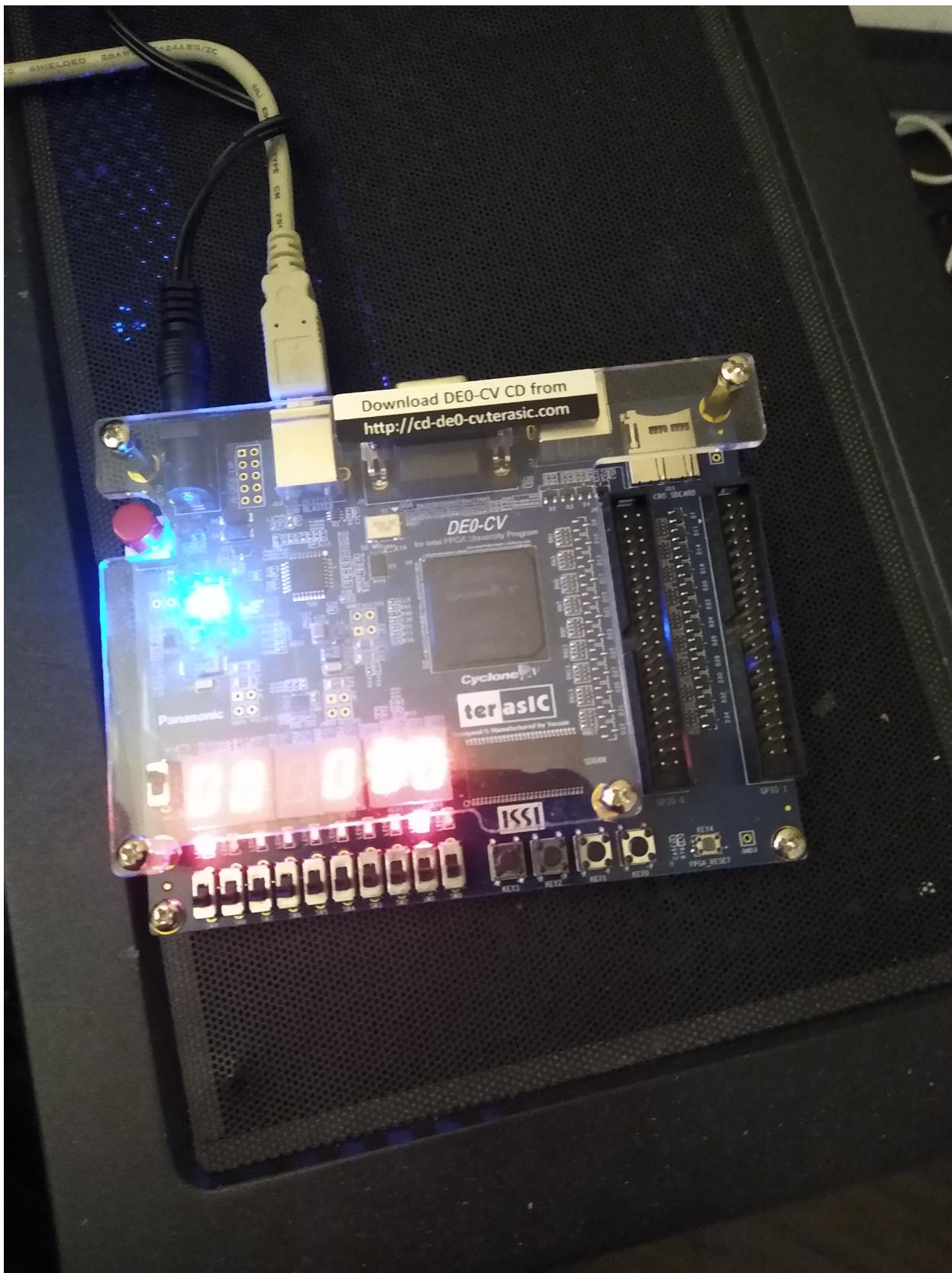


Figure 39: FPGA picture right after reset

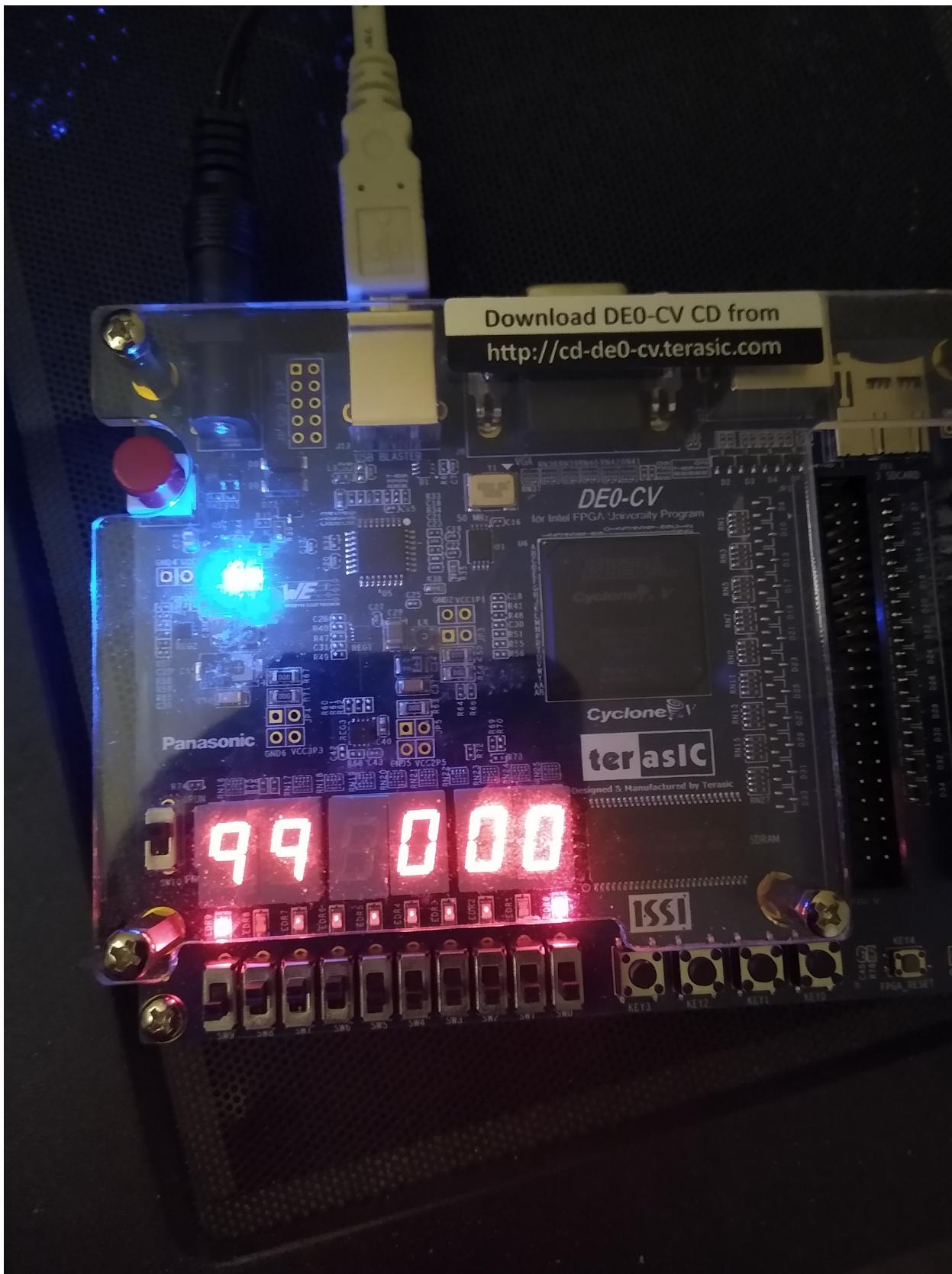


Figure 40: FPGA picture after logging in

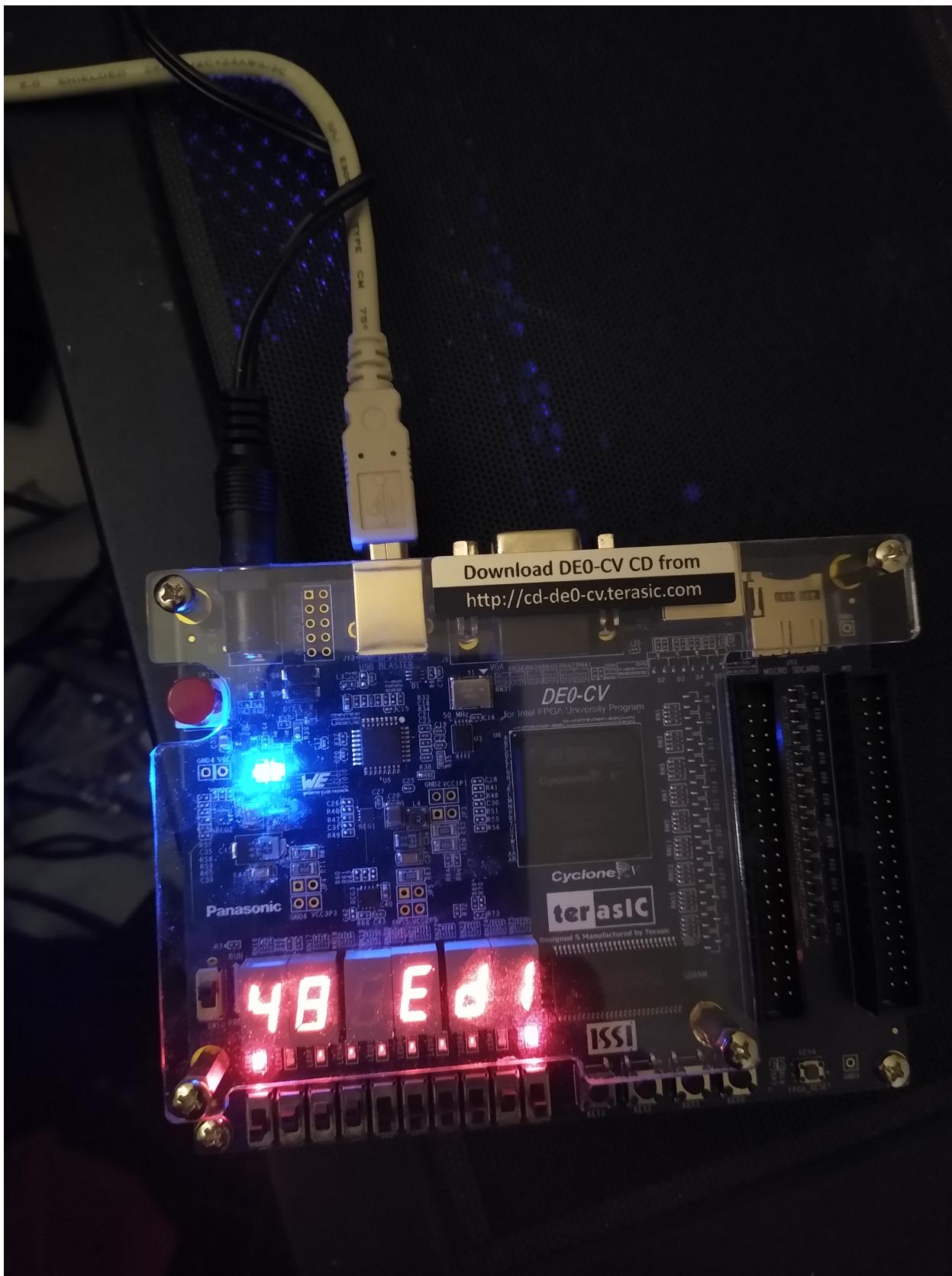


Figure 41: FPGA picture after hitting the random number generator button, entering in 0x1 for the player, and hitting the player load button

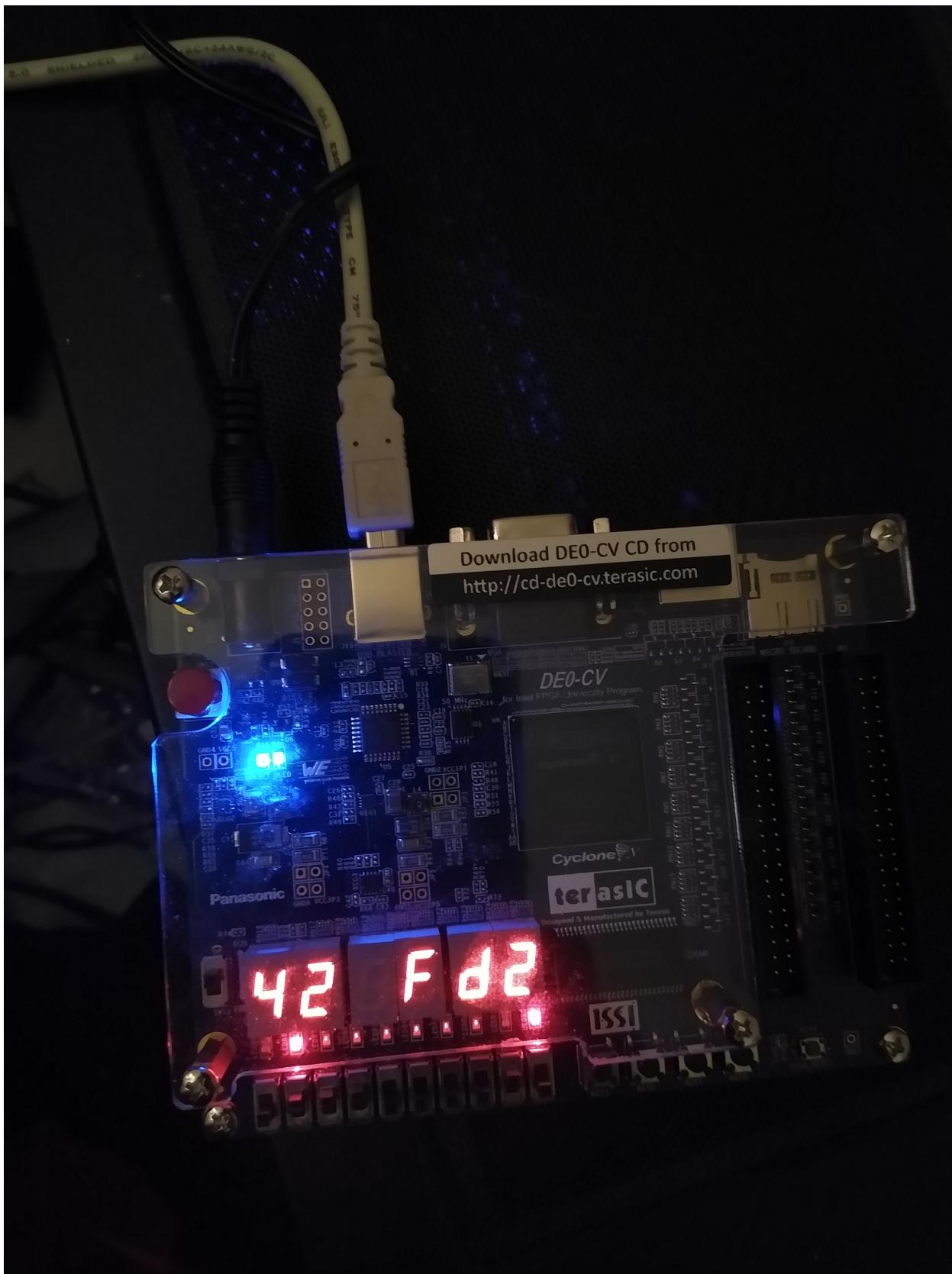


Figure 42: FPGA picture after entering in 0x2 for the player and hitting the player load button

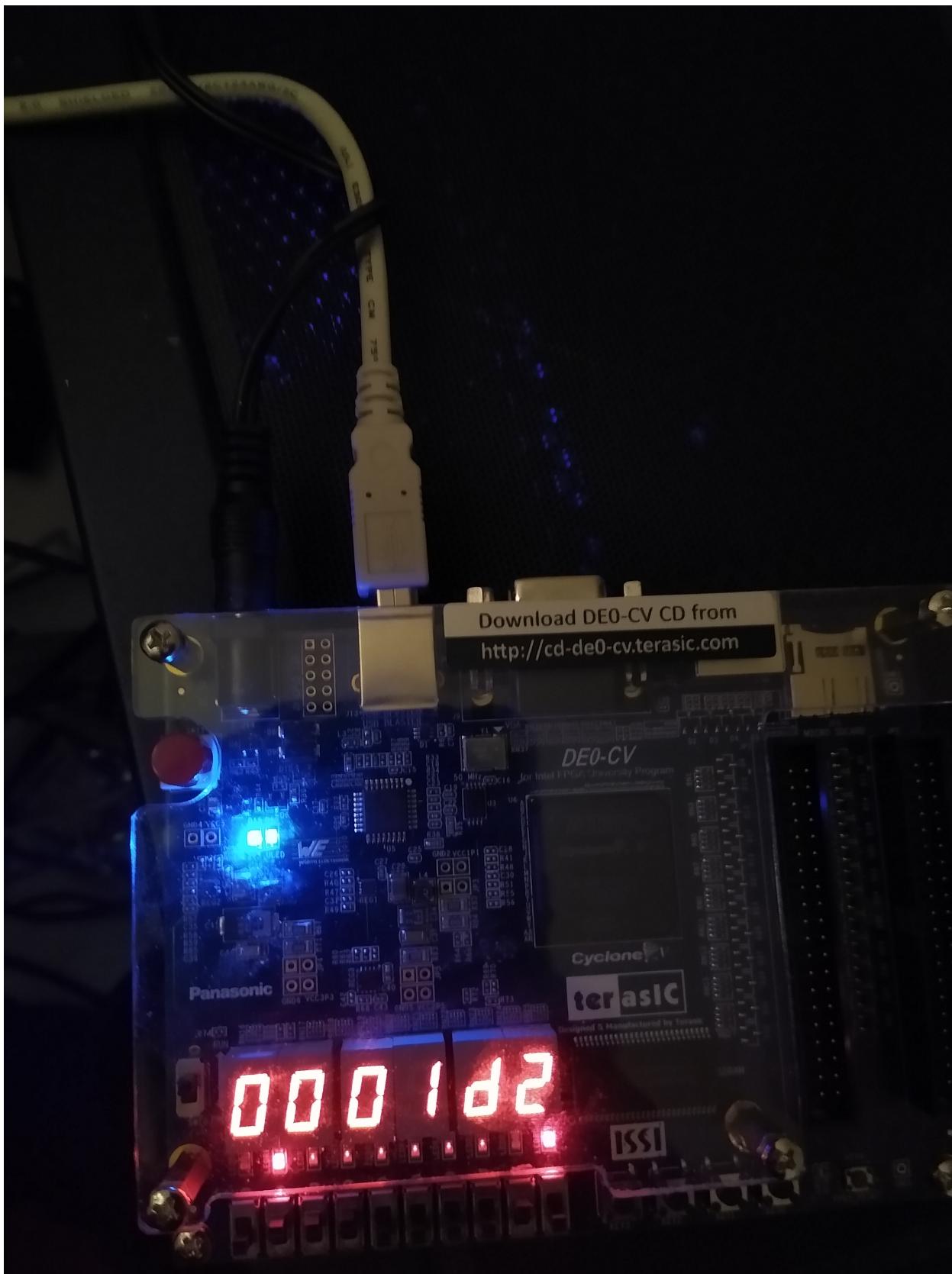


Figure 43: FPGA automatic scoring demo (bonus)

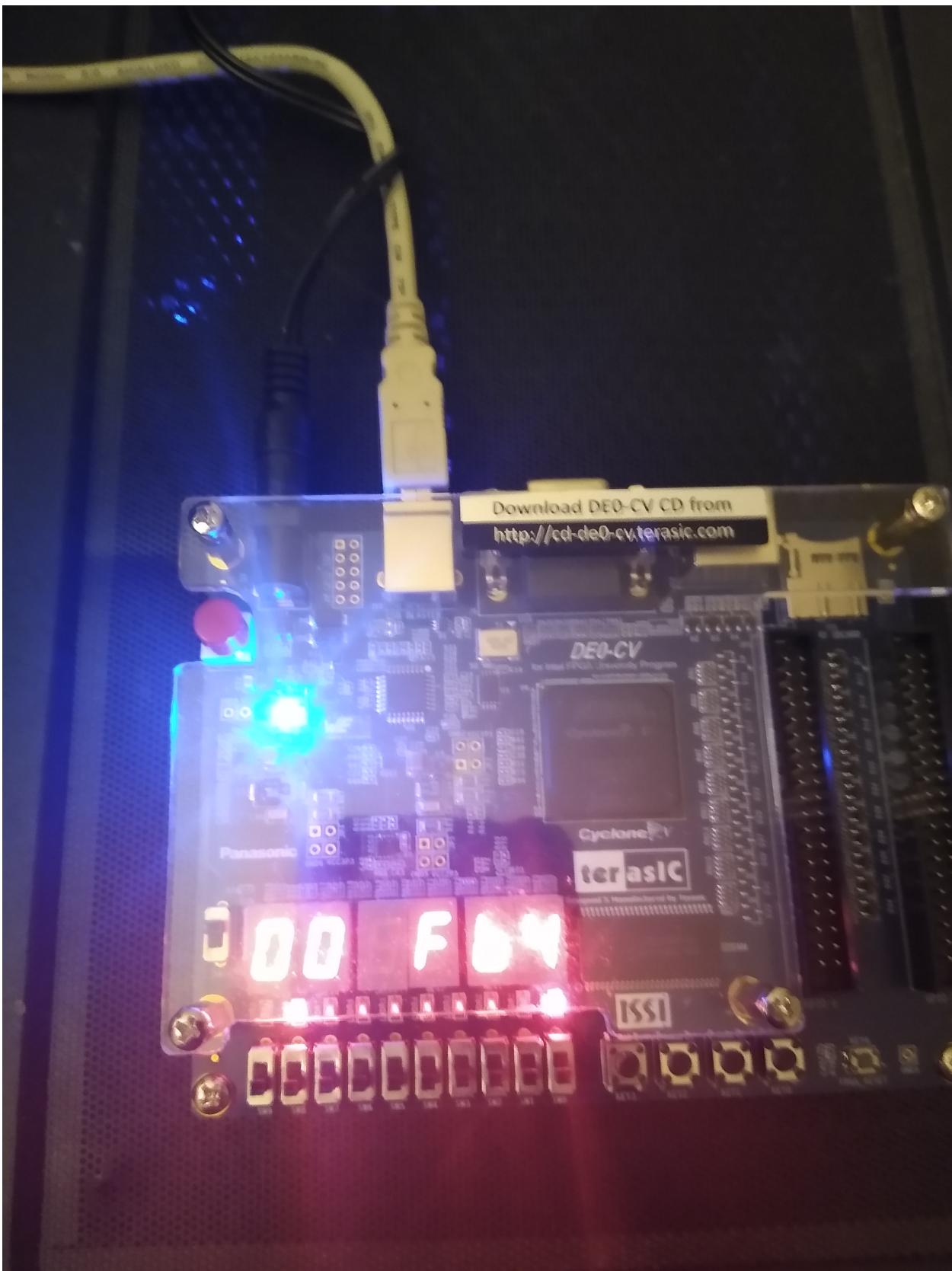


Figure 44: Password changing feature (bonus)

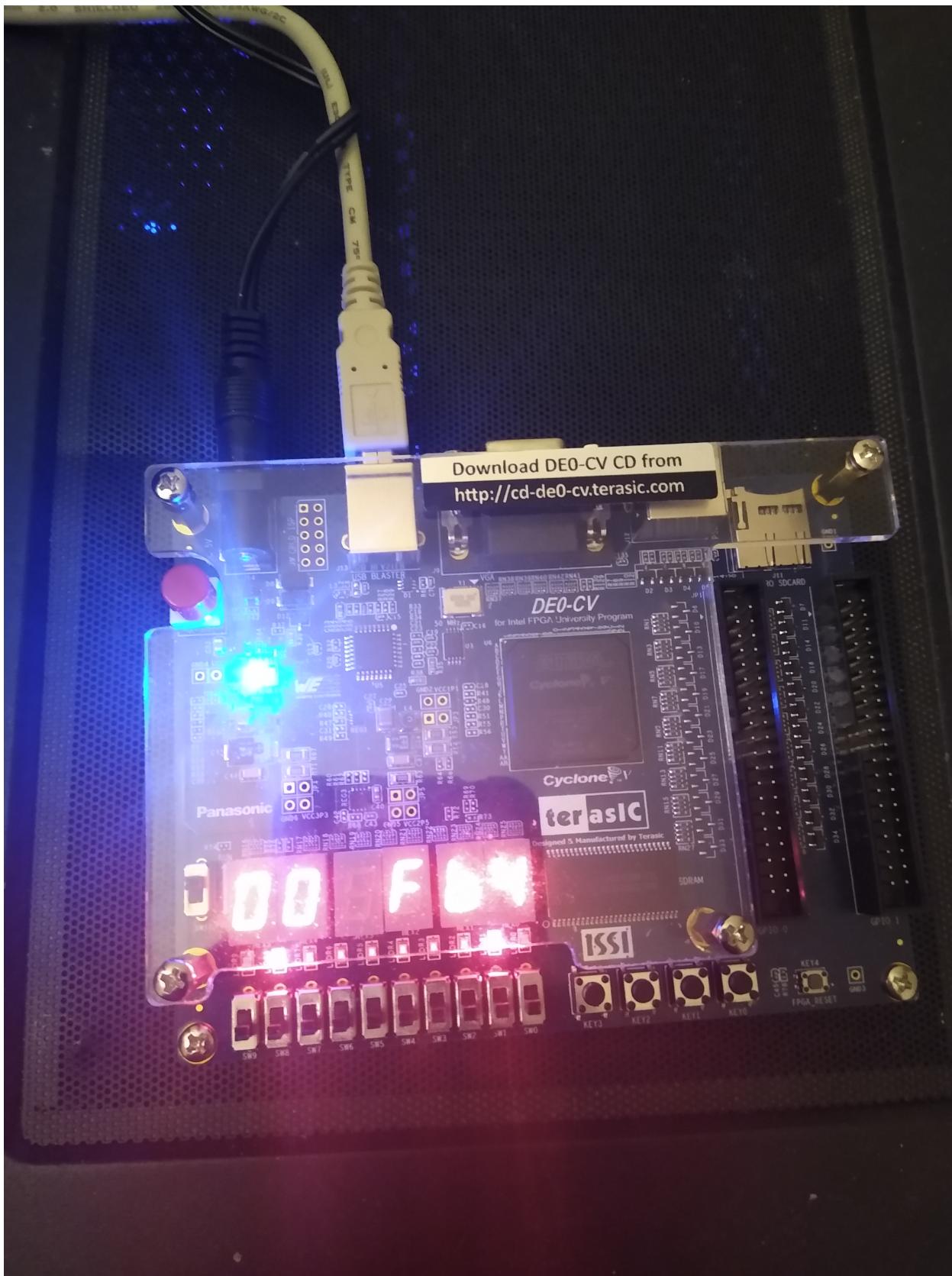


Figure 45: logout feature (bonus)

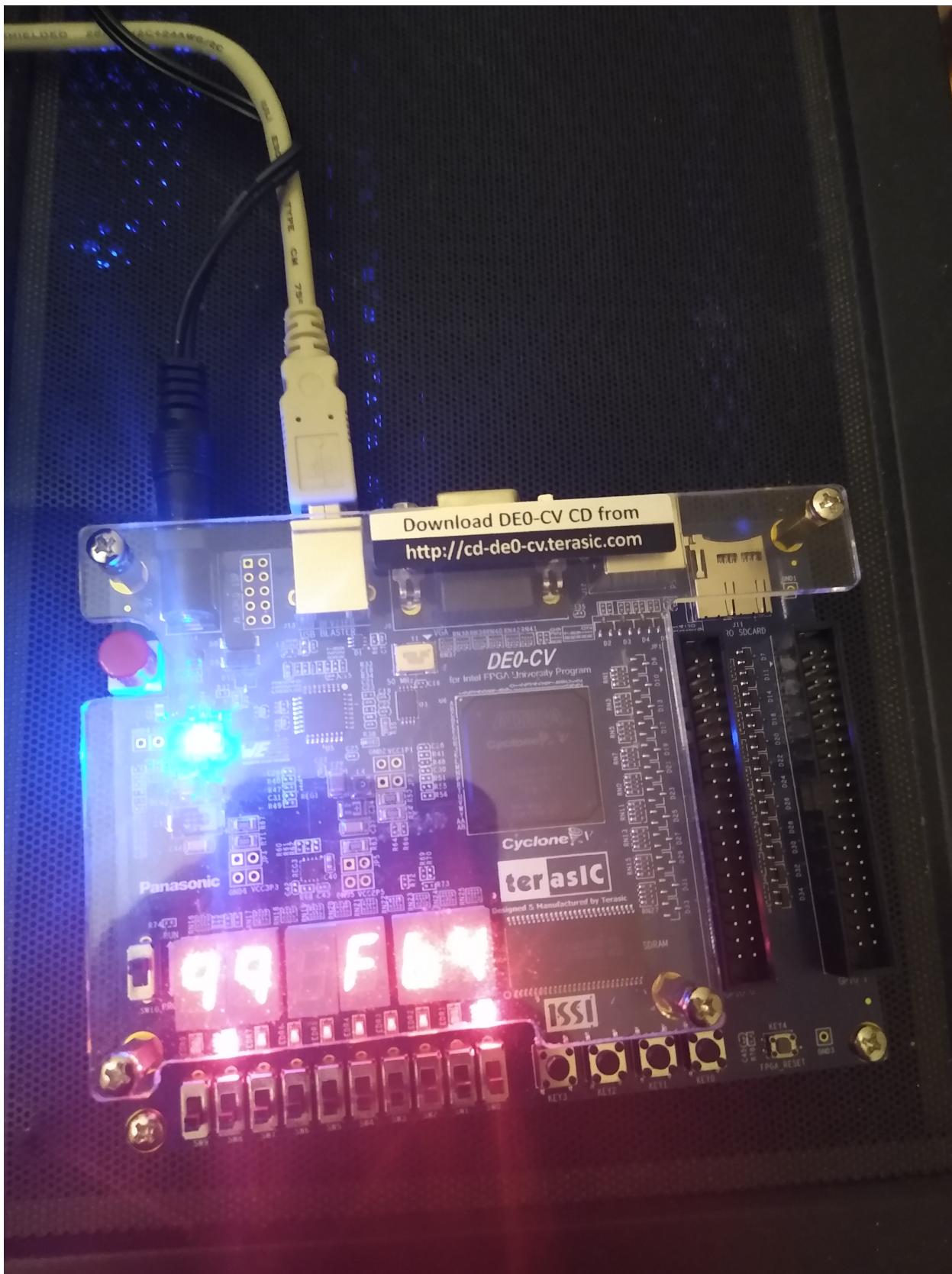


Figure 46: login after password reset

```
1 // C:\Users\Kiran\OneDrive\Documents\GitHub\Project\src\main.c
2 // (C) 2024
3 // Name: Kiran
4 // This module generates a random number and is LFSR based
5
6 module myModule_rv1(rv1, RND_Din, Random_num);
7   input clk;
8   input RND_Din;
9   output Random_num;
10
11   assign Random_num = lfsr_out;
12
13   reg [10] lfsr;
14   wire feedback = lfsr[1];
15   wire count;
16
17   assign count = COUNT[0];
18
19   assign Random_num = count_out;
20
21   always@posedge clk
22   begin
23     if(rnd == 1'b1)
24       count_out <= 4'b0000;
25     else
26       lfsr <= 10'b0000000000;
27   end
28   begin
29     if(count == 1'b1)
30       begin
31         lfsr[0] <= feedback;
32         lfsr[1] <= lfsr[0];
33       end
34     end
35   endmodule
```

Figure 47: LFSR Random number generator module Verilog code screen capture

Figure 48: game controller module screen capture



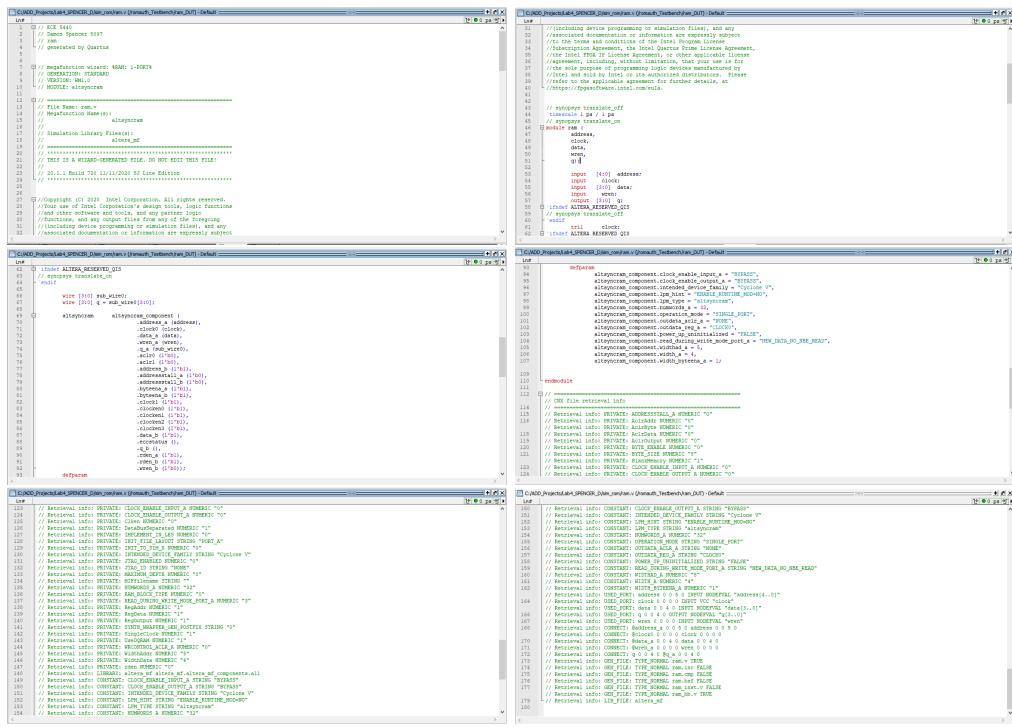


Figure 50: ram module code screen capture

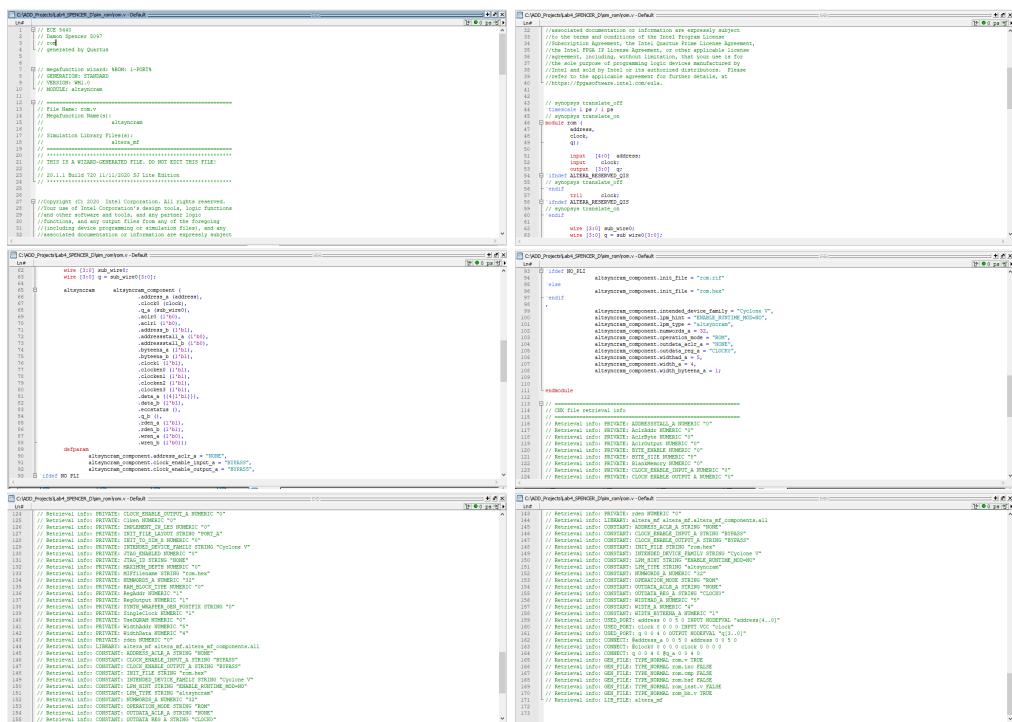


Figure 51: rom module code screen capture

The screenshot displays three separate windows of an IDE, likely QEMU, showing assembly code for a ROM test module. The code is identical across all three windows, demonstrating how the ROM is executed in memory.

```
1; tms320f28335_16M1000
2; @ C2K 9445
3; // Device Specific $01
4; // This file contains the assembly code for the rom module
5; // this module tests the rom module for correctness and allows me to figure out the correct timing
6;
7; #include <rom.h>
8; #include <romtest.h>
9; #include <syscalls.h>
10; #include <math.h>
11; #include <math.h>
12;
13; always
14; begin
15;   clk = '0';
16;   if (tms320f28335_16M1000 == 1)
17;     clk = '1';
18;   else
19;     clk = '0';
20;
21;   rom rom207(addr, clk, q_ROM);
22;
23;   initial
24;   begin
25;     addr = $100000;
26;     $readmemh("C:/Users/.../Desktop/rom207_hex_file/rom207_hex_file.rom",addr,q_ROM);
27;   end
28;   $display("q_ROM = %h",q_ROM);
29;   $display("addr = %h",addr);
30;   $display("clk = %b",clk);
31; end
```

Figure 52: rom module test bench Verilog code

Figure 53: rom and authentication module integration test bench Verilog code

Figure 54: onemslfsr module code screen capture

```

1 // timescale 1ns/100ps
2 module countTo100_Testbench();
3   reg clk;
4   reg rst;
5   reg enable_s;
6   wire countTo100_Timeout_s;
7
8   initial
9   begin
10    clk = 1'b0;
11    #10;
12    clk = 1'b1;
13    #10;
14  end
15  countTo100 countTo100_DUT(clk, rst, enable_s, countTo100_Timeout_s);
16
17 endmodule
18
19 initial
20 begin
21   rst = 1'b1;
22   enable_s = 1'b0;
23   #posedge clk;
24   #1;
25   clk = 1'b0;
26   #posedge clk;
27   #1;
28   clk = 1'b1;
29   #posedge clk;
30   #1;
31   enable_s = 1'b1;
32   #posedge clk;
33   #1;
34   #posedge clk;
35   #1;
36   #posedge clk;
37   #1;
38   #posedge clk;
39   #1;
40   #posedge clk;
41   #1;
42   #posedge clk;
43   #1;
44   #posedge clk;
45   #1;
46   #posedge clk;
47   #1;
48   #posedge clk;
49   #1;
50 end

```

Figure 55: 1 millisecond LFSR timer module Verilog code

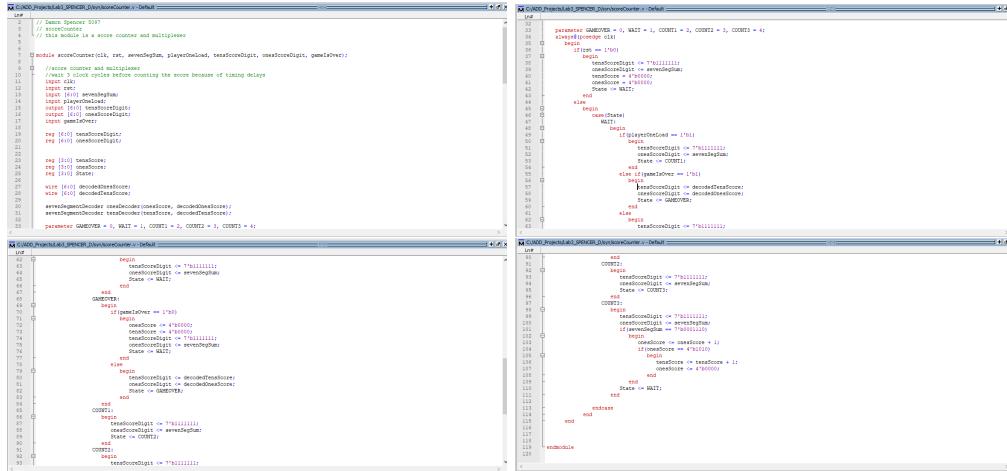


Figure 56: Score counter module Verilog code screen capture

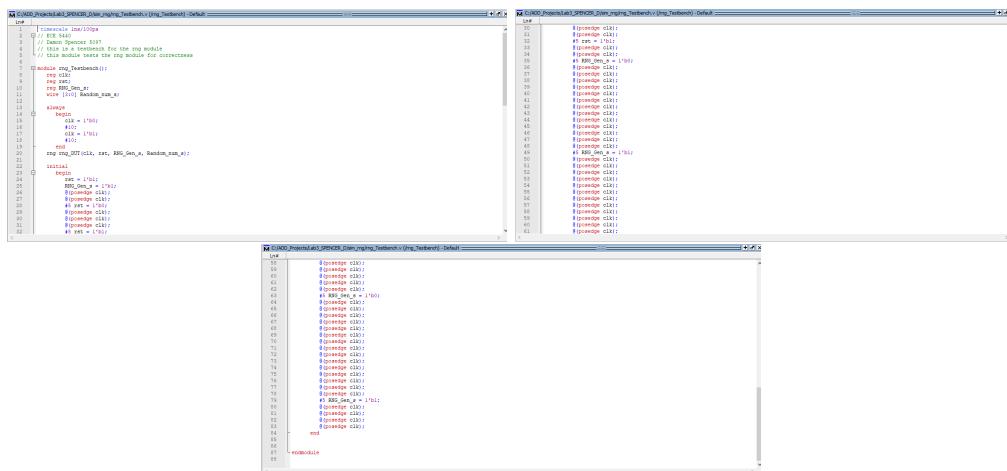


Figure 57: Random number generator test bench module Verilog code screen capture

```

module DigitTimer(Digit, DigitOut);
    input [7:0] Digit;
    output [7:0] DigitOut;
    reg [7:0] DigitOut;
    reg [3:0] Digit;
    input recodig;
    input [7:0] BorrowD;
    input [7:0] BorrowM;
    input [7:0] BorrowMD;
    input [7:0] BorrowMS;
    input [7:0] BorrowMDS;
    output [7:0] BorrowD;
    output [7:0] BorrowM;
    output [7:0] BorrowMD;
    output [7:0] BorrowMS;
    output [7:0] BorrowMDS;
endmodule

```

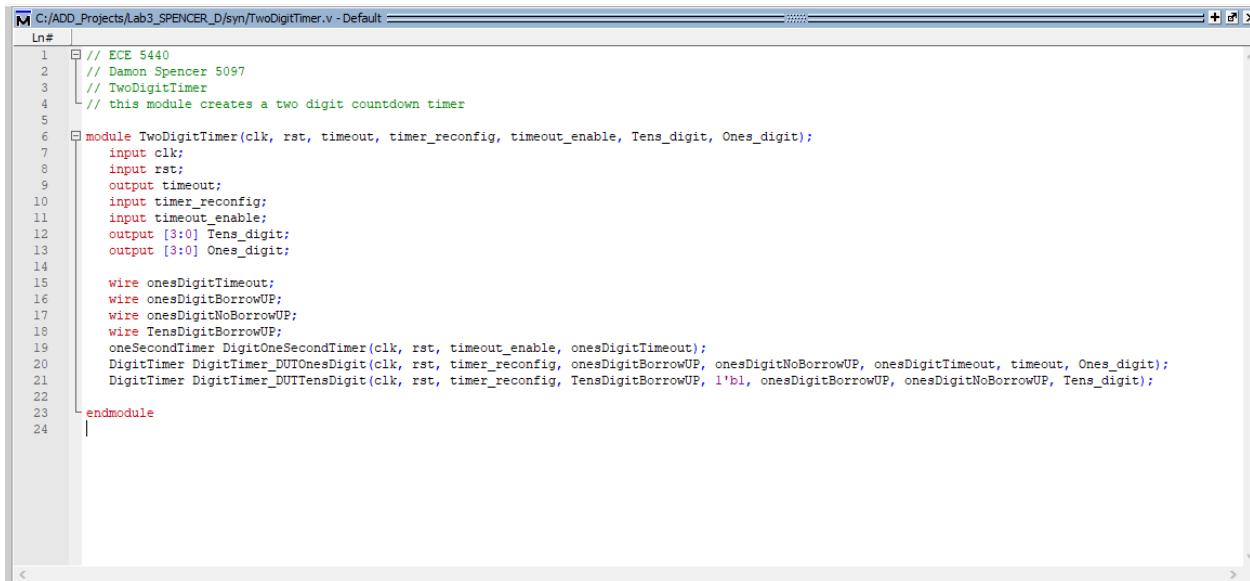
Figure 58: Digit timer module Verilog code screen capture

```

initial
begin
    $readmem("DigitTimes.dat", Digit);
    $readmem("DigitTimes.dat", BorrowD);
    $readmem("DigitTimes.dat", BorrowM);
    $readmem("DigitTimes.dat", BorrowMD);
    $readmem("DigitTimes.dat", BorrowMS);
    $readmem("DigitTimes.dat", BorrowMDS);
end

```

Figure 59: Digit timer module testbench Verilog code screen capture



The screenshot shows a Verilog code editor window titled "C:/ADD\_Projects/Lab3\_SPENCER\_D/syn/TwoDigitTimer.v - Default". The code is a Verilog module named "TwoDigitTimer". The module has inputs: clk, rst, timer\_reconfig, timeout\_enable, and outputs: Tens\_digit and Ones\_digit. It also includes internal wires for onesDigitTimeout, onesDigitBorrowUP, onesDigitNoBorrowUP, and TensDigitBorrowUP. The module is instantiated twice: oneSecondTimer and DigitTimer. The instantiation for oneSecondTimer is commented out. The instantiation for DigitTimer uses DUTOneDigit and DUTTensDigit components.

```
1 // ECE 5440
2 // Damon Spencer 5097
3 // TwoDigitTimer
4 // this module creates a two digit countdown timer
5
6 module TwoDigitTimer(clk, rst, timeout, timer_reconfig, timeout_enable, Tens_digit, Ones_digit);
7   input clk;
8   input rst;
9   output timeout;
10  input timer_reconfig;
11  input timeout_enable;
12  output [3:0] Tens_digit;
13  output [3:0] Ones_digit;
14
15  wire onesDigitTimeout;
16  wire onesDigitBorrowUP;
17  wire onesDigitNoBorrowUP;
18  wire TensDigitBorrowUP;
19  oneSecondTimer#(DUTOneDigit, DUTTensDigit) oneSecondTimer(.clk(clk), .rst(rst), .timeout(timeout), .timer_reconfig(timer_reconfig), .timeout_enable(timeout_enable), .Tens_digit(Tens_digit), .Ones_digit(Ones_digit));
20  DigitTimer#(DUTOneDigit, DUTTensDigit) DigitTimer(.clk(clk), .rst(rst), .timer_reconfig(timer_reconfig), .timeout(timeout), .timeout_enable(timeout_enable), .Tens_digit(Tens_digit), .Ones_digit(Ones_digit));
21
22
23
24 endmodule
```

Figure 60: Two Digit timer module Verilog code screen capture

```

1 // This is a testbench for the TwoDigitTimer module
2 // this is a testbench for the TwoDigitTimer module
3 // this module tests the TwoDigitTimer module for correctness
4
5
6 module TwoDigitTimer_Testbench();
7
8    reg [4:0] time_recodfig;
9    reg [4:0] time_timeout;
10   reg [4:0] time_stable;
11   reg [4:0] time;
12   reg [4:0] time_ms;
13
14   wire [timeout_A];
15   wire [10:0] One_Digit_W;
16
17   TwoDigitTimer TwoDigitTimer_01(time_recodfig, time_timeout, time_stable, time, One_Digit_W);
18
19   always
20   begin
21     time_recodfig = 1'000;
22     #100
23     time_recodfig = 1'000;
24     #100
25     time_recodfig = 1'000;
26   end
27
28   TwoDigitTimer_TwoDigitTimer_01.time_recodfig_A, time_timeout_A, time_stable_A, time_One_Digit_W, One_Digit_W;
29
30   initial
31   begin
32     time = 1'000;
33     time_ms = 1'000;
34   end
35
36   task read();
37     $readmem("C:/Users/ASUS/Desktop/TwoDigitTimer/TwoDigitTimer.sv", time_ms);
38   endtask
39
40   initial
41   begin
42     time_ms = 1'000;
43   end
44
45   initial
46   begin
47     time_ms = 1'000;
48   end
49
50   initial
51   begin
52     time_ms = 1'000;
53   end
54
55   initial
56   begin
57     time_ms = 1'000;
58   end
59
60   initial
61   begin
62     time_ms = 1'000;
63   end
64
65   initial
66   begin
67     time_ms = 1'000;
68   end
69
70   initial
71   begin
72     time_ms = 1'000;
73   end
74
75   initial
76   begin
77     time_ms = 1'000;
78   end
79
80   initial
81   begin
82     time_ms = 1'000;
83   end
84
85   initial
86   begin
87     time_ms = 1'000;
88   end
89
90   initial
91   begin
92     time_ms = 1'000;
93   end
94
95   initial
96   begin
97     time_ms = 1'000;
98   end
99
100  initial
101   begin
102     time_ms = 1'000;
103   end
104
105  initial
106   begin
107     time_ms = 1'000;
108   end
109
110  initial
111   begin
112     time_ms = 1'000;
113   end
114
115  initial
116   begin
117     time_ms = 1'000;
118   end
119
120  initial
121   begin
122     time_ms = 1'000;
123   end
124
125  initial
126   begin
127     time_ms = 1'000;
128   end
129
130  initial
131   begin
132     time_ms = 1'000;
133   end
134
135  initial
136   begin
137     time_ms = 1'000;
138   end
139
140  initial
141   begin
142     time_ms = 1'000;
143   end
144
145  initial
146   begin
147     time_ms = 1'000;
148   end
149
150  initial
151   begin
152     time_ms = 1'000;
153   end
154
155  initial
156   begin
157     time_ms = 1'000;
158   end
159
160  initial
161   begin
162     time_ms = 1'000;
163   end
164
165  initial
166   begin
167     time_ms = 1'000;
168   end
169
170  initial
171   begin
172     time_ms = 1'000;
173   end
174
175  initial
176   begin
177     time_ms = 1'000;
178   end
179
180  initial
181   begin
182     time_ms = 1'000;
183   end
184
185  initial
186   begin
187     time_ms = 1'000;
188   end
189
190  initial
191   begin
192     time_ms = 1'000;
193   end
194
195  initial
196   begin
197     time_ms = 1'000;
198   end
199
200  initial
201   begin
202     time_ms = 1'000;
203   end
204
205  initial
206   begin
207     time_ms = 1'000;
208   end
209
210  initial
211   begin
212     time_ms = 1'000;
213   end
214
215  initial
216   begin
217     time_ms = 1'000;
218   end
219
220  initial
221   begin
222     time_ms = 1'000;
223   end
224
225  initial
226   begin
227     time_ms = 1'000;
228   end
229
230  initial
231   begin
232     time_ms = 1'000;
233   end
234
235  initial
236   begin
237     time_ms = 1'000;
238   end
239
240  initial
241   begin
242     time_ms = 1'000;
243   end
244
245  initial
246   begin
247     time_ms = 1'000;
248   end
249
250  initial
251   begin
252     time_ms = 1'000;
253   end
254
255  initial
256   begin
257     time_ms = 1'000;
258   end
259
260  initial
261   begin
262     time_ms = 1'000;
263   end
264
265  initial
266   begin
267     time_ms = 1'000;
268   end
269
270  initial
271   begin
272     time_ms = 1'000;
273   end
274
275  initial
276   begin
277     time_ms = 1'000;
278   end
279
280  initial
281   begin
282     time_ms = 1'000;
283   end
284
285  initial
286   begin
287     time_ms = 1'000;
288   end
289
290  initial
291   begin
292     time_ms = 1'000;
293   end
294
295  initial
296   begin
297     time_ms = 1'000;
298   end
299
299  initial
300   begin
301     time_ms = 1'000;
302   end
303
304  initial
305   begin
306     time_ms = 1'000;
307   end
308
309  initial
310   begin
311     time_ms = 1'000;
312   end
313
314  initial
315   begin
316     time_ms = 1'000;
317   end
318
319  initial
320   begin
321     time_ms = 1'000;
322   end
323
324  initial
325   begin
326     time_ms = 1'000;
327   end
328
329  initial
330   begin
331     time_ms = 1'000;
332   end
333
334  initial
335   begin
336     time_ms = 1'000;
337   end
338
339  initial
340   begin
341     time_ms = 1'000;
342   end
343
344  initial
345   begin
346     time_ms = 1'000;
347   end
348
349  initial
350   begin
351     time_ms = 1'000;
352   end
353
354  initial
355   begin
356     time_ms = 1'000;
357   end
358
359  initial
360   begin
361     time_ms = 1'000;
362   end
363
364  initial
365   begin
366     time_ms = 1'000;
367   end
368
369  initial
370   begin
371     time_ms = 1'000;
372   end
373
374  initial
375   begin
376     time_ms = 1'000;
377   end
378
379  initial
380   begin
381     time_ms = 1'000;
382   end
383
384  initial
385   begin
386     time_ms = 1'000;
387   end
388
389  initial
390   begin
391     time_ms = 1'000;
392   end
393
394  initial
395   begin
396     time_ms = 1'000;
397   end
398
399  initial
400   begin
401     time_ms = 1'000;
402   end
403
404  initial
405   begin
406     time_ms = 1'000;
407   end
408
409  initial
410   begin
411     time_ms = 1'000;
412   end
413
414  initial
415   begin
416     time_ms = 1'000;
417   end
418
419  initial
420   begin
421     time_ms = 1'000;
422   end
423
424  initial
425   begin
426     time_ms = 1'000;
427   end
428
429  initial
430   begin
431     time_ms = 1'000;
432   end
433
434  initial
435   begin
436     time_ms = 1'000;
437   end
438
439  initial
440   begin
441     time_ms = 1'000;
442   end
443
444  initial
445   begin
446     time_ms = 1'000;
447   end
448
449  initial
450   begin
451     time_ms = 1'000;
452   end
453
454  initial
455   begin
456     time_ms = 1'000;
457   end
458
459  initial
460   begin
461     time_ms = 1'000;
462   end
463
464  initial
465   begin
466     time_ms = 1'000;
467   end
468
469  initial
470   begin
471     time_ms = 1'000;
472   end
473
474  initial
475   begin
476     time_ms = 1'000;
477   end
478
479  initial
480   begin
481     time_ms = 1'000;
482   end
483
484  initial
485   begin
486     time_ms = 1'000;
487   end
488
489  initial
490   begin
491     time_ms = 1'000;
492   end
493
494  initial
495   begin
496     time_ms = 1'000;
497   end
498
499  initial
500   begin
501     time_ms = 1'000;
502   end
503
504  initial
505   begin
506     time_ms = 1'000;
507   end
508
509  initial
510   begin
511     time_ms = 1'000;
512   end
513
514  initial
515   begin
516     time_ms = 1'000;
517   end
518
519  initial
520   begin
521     time_ms = 1'000;
522   end
523
524  initial
525   begin
526     time_ms = 1'000;
527   end
528
529  initial
530   begin
531     time_ms = 1'000;
532   end
533
534  initial
535   begin
536     time_ms = 1'000;
537   end
538
539  initial
540   begin
541     time_ms = 1'000;
542   end
543
544  initial
545   begin
546     time_ms = 1'000;
547   end
548
549  initial
550   begin
551     time_ms = 1'000;
552   end
553
554  initial
555   begin
556     time_ms = 1'000;
557   end
558
559  initial
560   begin
561     time_ms = 1'000;
562   end
563
564  initial
565   begin
566     time_ms = 1'000;
567   end
568
569  initial
570   begin
571     time_ms = 1'000;
572   end
573
574  initial
575   begin
576     time_ms = 1'000;
577   end
578
579  initial
580   begin
581     time_ms = 1'000;
582   end
583
584  initial
585   begin
586     time_ms = 1'000;
587   end
588
589  initial
590   begin
591     time_ms = 1'000;
592   end
593
594  initial
595   begin
596     time_ms = 1'000;
597   end
598
599  initial
600   begin
601     time_ms = 1'000;
602   end
603
604  initial
605   begin
606     time_ms = 1'000;
607   end
608
609  initial
610   begin
611     time_ms = 1'000;
612   end
613
614  initial
615   begin
616     time_ms = 1'000;
617   end
618
619  initial
620   begin
621     time_ms = 1'000;
622   end
623
624  initial
625   begin
626     time_ms = 1'000;
627   end
628
629  initial
630   begin
631     time_ms = 1'000;
632   end
633
634  initial
635   begin
636     time_ms = 1'000;
637   end
638
639  initial
640   begin
641     time_ms = 1'000;
642   end
643
644  initial
645   begin
646     time_ms = 1'000;
647   end
648
649  initial
650   begin
651     time_ms = 1'000;
652   end
653
654  initial
655   begin
656     time_ms = 1'000;
657   end
658
659  initial
660   begin
661     time_ms = 1'000;
662   end
663
664  initial
665   begin
666     time_ms = 1'000;
667   end
668
669  initial
670   begin
671     time_ms = 1'000;
672   end
673
674  initial
675   begin
676     time_ms = 1'000;
677   end
678
679  initial
680   begin
681     time_ms = 1'000;
682   end
683
684  initial
685   begin
686     time_ms = 1'000;
687   end
688
689  initial
690   begin
691     time_ms = 1'000;
692   end
693
694  initial
695   begin
696     time_ms = 1'000;
697   end
698
699  initial
700   begin
701     time_ms = 1'000;
702   end
703
704  initial
705   begin
706     time_ms = 1'000;
707   end
708
709  initial
710   begin
711     time_ms = 1'000;
712   end
713
714  initial
715   begin
716     time_ms = 1'000;
717   end
718
719  initial
720   begin
721     time_ms = 1'000;
722   end
723
724  initial
725   begin
726     time_ms = 1'000;
727   end
728
729  initial
730   begin
731     time_ms = 1'000;
732   end
733
734  initial
735   begin
736     time_ms = 1'000;
737   end
738
739  initial
740   begin
741     time_ms = 1'000;
742   end
743
744  initial
745   begin
746     time_ms = 1'000;
747   end
748
749  initial
750   begin
751     time_ms = 1'000;
752   end
753
754  initial
755   begin
756     time_ms = 1'000;
757   end
758
759  initial
760   begin
761     time_ms = 1'000;
762   end
763
764  initial
765   begin
766     time_ms = 1'000;
767   end
768
769  initial
770   begin
771     time_ms = 1'000;
772   end
773
774  initial
775   begin
776     time_ms = 1'000;
777   end
778
779  initial
780   begin
781     time_ms = 1'000;
782   end
783
784  initial
785   begin
786     time_ms = 1'000;
787   end
788
789  initial
790   begin
791     time_ms = 1'000;
792   end
793
794  initial
795   begin
796     time_ms = 1'000;
797   end
798
799  initial
800   begin
801     time_ms = 1'000;
802   end
803
804  initial
805   begin
806     time_ms = 1'000;
807   end
808
809  initial
810   begin
811     time_ms = 1'000;
812   end
813
814  initial
815   begin
816     time_ms = 1'000;
817   end
818
819  initial
820   begin
821     time_ms = 1'000;
822   end
823
824  initial
825   begin
826     time_ms = 1'000;
827   end
828
829  initial
830   begin
831     time_ms = 1'000;
832   end
833
834  initial
835   begin
836     time_ms = 1'000;
837   end
838
839  initial
840   begin
841     time_ms = 1'000;
842   end
843
844  initial
845   begin
846     time_ms = 1'000;
847   end
848
849  initial
850   begin
851     time_ms = 1'000;
852   end
853
854  initial
855   begin
856     time_ms = 1'000;
857   end
858
859  initial
860   begin
861     time_ms = 1'000;
862   end
863
864  initial
865   begin
866     time_ms = 1'000;
867   end
868
869  initial
870   begin
871     time_ms = 1'000;
872   end
873
874  initial
875   begin
876     time_ms = 1'000;
877   end
878
879  initial
880   begin
881     time_ms = 1'000;
882   end
883
884  initial
885   begin
886     time_ms = 1'000;
887   end
888
889  initial
890   begin
891     time_ms = 1'000;
892   end
893
894  initial
895   begin
896     time_ms = 1'000;
897   end
898
899  initial
900   begin
901     time_ms = 1'000;
902   end
903
904  initial
905   begin
906     time_ms = 1'000;
907   end
908
909  initial
910   begin
911     time_ms = 1'000;
912   end
913
914  initial
915   begin
916     time_ms = 1'000;
917   end
918
919  initial
920   begin
921     time_ms = 1'000;
922   end
923
924  initial
925   begin
926     time_ms = 1'000;
927   end
928
929  initial
930   begin
931     time_ms = 1'000;
932   end
933
934  initial
935   begin
936     time_ms = 1'000;
937   end
938
939  initial
940   begin
941     time_ms = 1'000;
942   end
943
944  initial
945   begin
946     time_ms = 1'000;
947   end
948
949  initial
950   begin
951     time_ms = 1'000;
952   end
953
954  initial
955   begin
956     time_ms = 1'000;
957   end
958
959  initial
960   begin
961     time_ms = 1'000;
962   end
963
964  initial
965   begin
966     time_ms = 1'000;
967   end
968
969  initial
970   begin
971     time_ms = 1'000;
972   end
973
974  initial
975   begin
976     time_ms = 1'000;
977   end
978
979  initial
980   begin
981     time_ms = 1'000;
982   end
983
984  initial
985   begin
986     time_ms = 1'000;
987   end
988
989  initial
990   begin
991     time_ms = 1'000;
992   end
993
994  initial
995   begin
996     time_ms = 1'000;
997   end
998
999  initial
1000 begin
1001   time_ms = 1'000;
1002 end
1003
1004 endmodule

```

Figure 61: Two Digit timer module testbench Verilog code screen capture

```

1  module oneSecondTimer(clk, rst, enable, oneSecond_Timeout);
2      input clk;
3      input rst;
4      input enable;
5      output oneSecond_Timeout;
6      wire oneHundredms_Timeout;
7      countTo10 to10Counter(clk, rst, oneHundredms_Timeout, oneSecond_Timeout);
8      oneHundredmsTimer oneHundredmsCounter(clk, rst, enable, oneHundredms_Timeout);
9  endmodule
10

```

Figure 62: oneSecondTimer module Verilog code screen capture

```

1  'timescale 1ns/100ps
2  module countTo100_Testbench();
3      reg rst;
4      reg enable_s;
5      wire countTo100_Timeout_s;
6
7      always begin
8          clk = 1'b0;
9          #10;
10         countTo100 countTo100_DUT(clk, rst, enable_s, countTo100_Timeout_s);
11     end
12
13     initial begin
14         rst = 1'b1;
15         enable_s = 1'b0;
16         #10;
17         $initial
18         begin
19             rst = 1'b0;
20             enable_s = 1'b0;
21             #10;
22             $posedge clk;
23             #10;
24             #!rst = 1'b0;
25             #10;
26             $posedge clk;
27             #10;
28             $posedge clk;
29             #10;
30             $posedge clk;
31             #10;
32             $posedge clk;
33             #10;
34             $posedge clk;
35             #10;
36             $posedge clk;
37             #10;
38             $posedge clk;
39             #10;
40             $posedge clk;
41             #10;
42             $posedge clk;
43             #10;
44             $posedge clk;
45             #10;
46             $posedge clk;
47             #10;
48             $posedge clk;
49             #10;
50         end
51     endmodule

```

Figure 63: oneSecondTimer test bench module code screen capture

```

1  module oneSecondTimer(clk, rst, enable, oneSecond_Timeout);
2      input clk;
3      input rst;
4      input enable;
5      output oneSecond_Timeout;
6      wire oneHundredms_Timeout;
7      countTo10 to10Counter(clk, rst, oneHundredms_Timeout, oneSecond_Timeout);
8      oneHundredmsTimer oneHundredmsCounter(clk, rst, enable, oneHundredms_Timeout);
9  endmodule
10

```

Figure 64: oneHundredmsTimer module code screen capture

```
1  module countTo10(clk,rst,enable, countTo10_Timeout);
2      input clk;
3      input rst;
4      input enable;
5      output countTo10_Timeout;
6
7      reg [3:0] count;
8      reg countTo10_Timeout;
9      always @(posedge clk)
10     begin
11         if(rst == 1'b0)
12             begin
13                 count <= 16'b0000;
14                 countTo10_Timeout <= 1'b0;
15             end
16         else
17             begin
18                 if(enable == 1'b1)
19                     begin
20                         if(count == 1) //need to only count to 1 to have the timer go off every other cycle
21                             begin
22                                 count <= 16'b0000;
23                                 countTo10_Timeout <= 1'b1;
24                             end
25                         else
26                             begin
27                                 count <= count + 1;
28                                 countTo10_Timeout <= 1'b0;
29                             end
30                         end
31                     else
32                         begin
33                             countTo10_Timeout <= 1'b0;
34                         end
35                 end
36             end
37         end
38
39     endmodule
40
```

Figure 65: countTo10 module code screen capture

```
1  module countTo100(clk,rst,enable, countTo100_Timeout);
2      input clk;
3      input rst;
4      input enable;
5      output countTo100_Timeout;
6
7      reg [6:0] count;
8      reg countTo100_Timeout;
9      always @(posedge clk)
10         begin
11             if(rst == 1'b0)
12                 begin
13                     count <= 16'b0000;
14                     countTo100_Timeout <= 1'b0;
15                 end
16             else
17                 begin
18                     if(enable == 1'bl)
19                         begin
20                             if(count == 2) //need to only count to 2 to have the timer go off every three cycles
21                             begin
22                                 count <= 16'b0000;
23                                 countTo100_Timeout <= 1'bl;
24                             end
25                         else
26                             begin
27                                 count <= count + 1;
28                                 countTo100_Timeout <= 1'b0;
29                             end
30                         end
31                     else
32                         begin
33                             countTo100_Timeout <= 1'b0;
34                         end
35                 end
36             end
37         end
38     endmodule
39
40
```

Figure 66: countTo100 module code screen capture

```

Ln# 1 module LoadRegister(D_in, D_out, clk, rst, Load);
2   input [3:0] D_in;
3   output [3:0] D_out;
4   input clk, rst;
5   input Load;
6   reg [3:0] D_out;
7
8   always@(posedge clk)
9     begin
10       if(rst == 1'b0)
11         begin
12           D_out <= 4'b0000;
13         end
14       else
15         //normal operation after reset
16         begin
17           if(Load == 1'b1)
18             begin
19               D_out <= D_in;
20             end
21         end
22     end
23
24
25   endmodule
26

```

Figure 67: Load register module Verilog code screen capture

```

Ln# 1 module tb_loadregister();
2   timecale 1ns/1ps;
3
4   module testbench_LoadRegister();
5     reg rst, Load;
6     reg [3:0] D_in;
7     wire [3:0] D_out;
8
9     //Params
10    parameter
11      initial
12        D_in = 1'b0;
13        Load = 1'b0;
14
15    end
16
17    LoadRegister DUT_LoadRegister(D_in, D_out, clk, rst, Load);
18
19    initial
20      begin
21        Load = 1'b0;
22        D_in = 1'b0;
23        #10ns
24        $posedge clk;
25        $display("Initial State: D_out = %b", D_out);
26        $display("Initial State: Load = %b", Load);
27        $display("Initial State: D_in = %b", D_in);
28
29        #10ns
30        Load = 1'b1;
31        $posedge clk;
32        $display("Load Set to 1'b1");
33        $display("Initial State: D_out = %b", D_out);
34        $display("Initial State: Load = %b", Load);
35        $display("Initial State: D_in = %b", D_in);
36
37        #10ns
38        Load = 1'b0;
39        $posedge clk;
40        $display("Load Set to 1'b0");
41        $display("Initial State: D_out = %b", D_out);
42        $display("Initial State: Load = %b", Load);
43        $display("Initial State: D_in = %b", D_in);
44
45        #10ns
46        Load = 1'b1;
47        $posedge clk;
48        $display("Load Set to 1'b1");
49        $display("Initial State: D_out = %b", D_out);
50        $display("Initial State: Load = %b", Load);
51
52    end
53
54  endmodule

```

Figure 68: Load register test bench module Verilog code screen capture

```

Ln# 1 module button_shaper(button_in, button_out, clk, RST);
2   parameter
3     localparam
4     button_in;
5     button_out;
6     clk;
7     RST;
8
9   localparam
10    state;
11    state1;
12    state2;
13
14    state = 1'b0;
15    state1 = 1'b0;
16    state2 = 1'b0;
17
18    state1 = state;
19    state2 = state;
20
21    state1 = state;
22    state2 = state;
23
24    state1 = state;
25    state2 = state;
26
27    state1 = state;
28    state2 = state;
29
30    state1 = state;
31    state2 = state;
32
33    state1 = state;
34    state2 = state;
35
36    state1 = state;
37    state2 = state;
38
39    state1 = state;
40    state2 = state;
41
42    state1 = state;
43    state2 = state;
44
45    state1 = state;
46    state2 = state;
47
48    state1 = state;
49    state2 = state;
50
51    state1 = state;
52    state2 = state;
53
54  endmodule

```

Figure 69: Button shaper module Verilog code screen capture

```

1 //verilog
2
3 module testbench_buttonshaper();
4
5   reg [3:0] button_in_a;
6   reg [3:0] button_in_b;
7   reg [3:0] button_in_c;
8
9   wire [3:0] button_out_a;
10  wire [3:0] button_out_b;
11  wire [3:0] button_out_c;
12
13  reg [3:0] cik;
14
15  reg [3:0] reti;
16
17  reg [3:0] button_in_d;
18
19  reg [3:0] button_out_d;
20
21  reg [3:0] button_in_e;
22
23  reg [3:0] button_out_e;
24
25  reg [3:0] button_in_f;
26
27  reg [3:0] button_out_f;
28
29  reg [3:0] button_in_g;
30
31  reg [3:0] button_out_g;
32
33  reg [3:0] button_in_h;
34
35  reg [3:0] button_out_h;
36
37  reg [3:0] button_in_i;
38
39  reg [3:0] button_out_i;
40
41
42  initial
43   begin
44     $monitor("%d", button_in_a);
45     $monitor("%d", button_in_b);
46     $monitor("%d", button_in_c);
47
48     $monitor("%d", button_out_a);
49     $monitor("%d", button_out_b);
50     $monitor("%d", button_out_c);
51
52     $monitor("%d", button_out_d);
53     $monitor("%d", button_out_e);
54     $monitor("%d", button_out_f);
55
56     $monitor("%d", button_out_g);
57     $monitor("%d", button_out_h);
58
59     $monitor("%d", button_out_i);
60
61     $monitor("%d", cik);
62
63     $monitor("%d", reti);
64
65   end
66
67
68  initial
69   begin
70     button_in_a = 4'b0000;
71     button_in_b = 4'b0001;
72     button_in_c = 4'b0010;
73     button_in_d = 4'b0011;
74     button_in_e = 4'b0100;
75     button_in_f = 4'b0101;
76     button_in_g = 4'b0110;
77     button_in_h = 4'b0111;
78     button_in_i = 4'b1000;
79
80     button_out_a = 4'b0000;
81     button_out_b = 4'b0001;
82     button_out_c = 4'b0010;
83     button_out_d = 4'b0011;
84     button_out_e = 4'b0100;
85     button_out_f = 4'b0101;
86     button_out_g = 4'b0110;
87     button_out_h = 4'b0111;
88     button_out_i = 4'b1000;
89
90     cik = 4'b0000;
91
92     reti = 4'b0000;
93
94   end
95
96
97  initial
98   begin
99     button_in_a = 4'b0000;
100    button_in_b = 4'b0001;
101    button_in_c = 4'b0010;
102    button_in_d = 4'b0011;
103    button_in_e = 4'b0100;
104    button_in_f = 4'b0101;
105    button_in_g = 4'b0110;
106    button_in_h = 4'b0111;
107    button_in_i = 4'b1000;
108
109    button_out_a = 4'b0000;
110    button_out_b = 4'b0001;
111    button_out_c = 4'b0010;
112    button_out_d = 4'b0011;
113    button_out_e = 4'b0100;
114    button_out_f = 4'b0101;
115    button_out_g = 4'b0110;
116    button_out_h = 4'b0111;
117    button_out_i = 4'b1000;
118
119    cik = 4'b0000;
120
121    reti = 4'b0000;
122
123  end
124
125
126  initial
127   begin
128     button_in_a = 4'b0000;
129     button_in_b = 4'b0001;
130     button_in_c = 4'b0010;
131     button_in_d = 4'b0011;
132     button_in_e = 4'b0100;
133     button_in_f = 4'b0101;
134     button_in_g = 4'b0110;
135     button_in_h = 4'b0111;
136     button_in_i = 4'b1000;
137
138     button_out_a = 4'b0000;
139     button_out_b = 4'b0001;
140     button_out_c = 4'b0010;
141     button_out_d = 4'b0011;
142     button_out_e = 4'b0100;
143     button_out_f = 4'b0101;
144     button_out_g = 4'b0110;
145     button_out_h = 4'b0111;
146     button_out_i = 4'b1000;
147
148     cik = 4'b0000;
149
150     reti = 4'b0000;
151
152  end
153
154
155  initial
156   begin
157     button_in_a = 4'b0000;
158     button_in_b = 4'b0001;
159     button_in_c = 4'b0010;
160     button_in_d = 4'b0011;
161     button_in_e = 4'b0100;
162     button_in_f = 4'b0101;
163     button_in_g = 4'b0110;
164     button_in_h = 4'b0111;
165     button_in_i = 4'b1000;
166
167     button_out_a = 4'b0000;
168     button_out_b = 4'b0001;
169     button_out_c = 4'b0010;
170     button_out_d = 4'b0011;
171     button_out_e = 4'b0100;
172     button_out_f = 4'b0101;
173     button_out_g = 4'b0110;
174     button_out_h = 4'b0111;
175     button_out_i = 4'b1000;
176
177     cik = 4'b0000;
178
179     reti = 4'b0000;
180
181  end
182
183
184  initial
185   begin
186     button_in_a = 4'b0000;
187     button_in_b = 4'b0001;
188     button_in_c = 4'b0010;
189     button_in_d = 4'b0011;
190     button_in_e = 4'b0100;
191     button_in_f = 4'b0101;
192     button_in_g = 4'b0110;
193     button_in_h = 4'b0111;
194     button_in_i = 4'b1000;
195
196     button_out_a = 4'b0000;
197     button_out_b = 4'b0001;
198     button_out_c = 4'b0010;
199     button_out_d = 4'b0011;
200     button_out_e = 4'b0100;
201     button_out_f = 4'b0101;
202     button_out_g = 4'b0110;
203     button_out_h = 4'b0111;
204     button_out_i = 4'b1000;
205
206     cik = 4'b0000;
207
208     reti = 4'b0000;
209
210  end
211
212
213  initial
214   begin
215     button_in_a = 4'b0000;
216     button_in_b = 4'b0001;
217     button_in_c = 4'b0010;
218     button_in_d = 4'b0011;
219     button_in_e = 4'b0100;
220     button_in_f = 4'b0101;
221     button_in_g = 4'b0110;
222     button_in_h = 4'b0111;
223     button_in_i = 4'b1000;
224
225     button_out_a = 4'b0000;
226     button_out_b = 4'b0001;
227     button_out_c = 4'b0010;
228     button_out_d = 4'b0011;
229     button_out_e = 4'b0100;
230     button_out_f = 4'b0101;
231     button_out_g = 4'b0110;
232     button_out_h = 4'b0111;
233     button_out_i = 4'b1000;
234
235     cik = 4'b0000;
236
237     reti = 4'b0000;
238
239  end
240
241
242  initial
243   begin
244     button_in_a = 4'b0000;
245     button_in_b = 4'b0001;
246     button_in_c = 4'b0010;
247     button_in_d = 4'b0011;
248     button_in_e = 4'b0100;
249     button_in_f = 4'b0101;
250     button_in_g = 4'b0110;
251     button_in_h = 4'b0111;
252     button_in_i = 4'b1000;
253
254     button_out_a = 4'b0000;
255     button_out_b = 4'b0001;
256     button_out_c = 4'b0010;
257     button_out_d = 4'b0011;
258     button_out_e = 4'b0100;
259     button_out_f = 4'b0101;
260     button_out_g = 4'b0110;
261     button_out_h = 4'b0111;
262     button_out_i = 4'b1000;
263
264     cik = 4'b0000;
265
266     reti = 4'b0000;
267
268  end
269
270
271  initial
272   begin
273     button_in_a = 4'b0000;
274     button_in_b = 4'b0001;
275     button_in_c = 4'b0010;
276     button_in_d = 4'b0011;
277     button_in_e = 4'b0100;
278     button_in_f = 4'b0101;
279     button_in_g = 4'b0110;
280     button_in_h = 4'b0111;
281     button_in_i = 4'b1000;
282
283     button_out_a = 4'b0000;
284     button_out_b = 4'b0001;
285     button_out_c = 4'b0010;
286     button_out_d = 4'b0011;
287     button_out_e = 4'b0100;
288     button_out_f = 4'b0101;
289     button_out_g = 4'b0110;
290     button_out_h = 4'b0111;
291     button_out_i = 4'b1000;
292
293     cik = 4'b0000;
294
295     reti = 4'b0000;
296
297  end
298
299
300  initial
301   begin
302     button_in_a = 4'b0000;
303     button_in_b = 4'b0001;
304     button_in_c = 4'b0010;
305     button_in_d = 4'b0011;
306     button_in_e = 4'b0100;
307     button_in_f = 4'b0101;
308     button_in_g = 4'b0110;
309     button_in_h = 4'b0111;
310     button_in_i = 4'b1000;
311
312     button_out_a = 4'b0000;
313     button_out_b = 4'b0001;
314     button_out_c = 4'b0010;
315     button_out_d = 4'b0011;
316     button_out_e = 4'b0100;
317     button_out_f = 4'b0101;
318     button_out_g = 4'b0110;
319     button_out_h = 4'b0111;
320     button_out_i = 4'b1000;
321
322     cik = 4'b0000;
323
324     reti = 4'b0000;
325
326  end
327
328
329  initial
330   begin
331     button_in_a = 4'b0000;
332     button_in_b = 4'b0001;
333     button_in_c = 4'b0010;
334     button_in_d = 4'b0011;
335     button_in_e = 4'b0100;
336     button_in_f = 4'b0101;
337     button_in_g = 4'b0110;
338     button_in_h = 4'b0111;
339     button_in_i = 4'b1000;
340
341     button_out_a = 4'b0000;
342     button_out_b = 4'b0001;
343     button_out_c = 4'b0010;
344     button_out_d = 4'b0011;
345     button_out_e = 4'b0100;
346     button_out_f = 4'b0101;
347     button_out_g = 4'b0110;
348     button_out_h = 4'b0111;
349     button_out_i = 4'b1000;
350
351     cik = 4'b0000;
352
353     reti = 4'b0000;
354
355  end
356
357
358  initial
359   begin
360     button_in_a = 4'b0000;
361     button_in_b = 4'b0001;
362     button_in_c = 4'b0010;
363     button_in_d = 4'b0011;
364     button_in_e = 4'b0100;
365     button_in_f = 4'b0101;
366     button_in_g = 4'b0110;
367     button_in_h = 4'b0111;
368     button_in_i = 4'b1000;
369
370     button_out_a = 4'b0000;
371     button_out_b = 4'b0001;
372     button_out_c = 4'b0010;
373     button_out_d = 4'b0011;
374     button_out_e = 4'b0100;
375     button_out_f = 4'b0101;
376     button_out_g = 4'b0110;
377     button_out_h = 4'b0111;
378     button_out_i = 4'b1000;
379
380     cik = 4'b0000;
381
382     reti = 4'b0000;
383
384  end
385
386
387  initial
388   begin
389     button_in_a = 4'b0000;
390     button_in_b = 4'b0001;
391     button_in_c = 4'b0010;
392     button_in_d = 4'b0011;
393     button_in_e = 4'b0100;
394     button_in_f = 4'b0101;
395     button_in_g = 4'b0110;
396     button_in_h = 4'b0111;
397     button_in_i = 4'b1000;
398
399     button_out_a = 4'b0000;
400     button_out_b = 4'b0001;
401     button_out_c = 4'b0010;
402     button_out_d = 4'b0011;
403     button_out_e = 4'b0100;
404     button_out_f = 4'b0101;
405     button_out_g = 4'b0110;
406     button_out_h = 4'b0111;
407     button_out_i = 4'b1000;
408
409     cik = 4'b0000;
410
411     reti = 4'b0000;
412
413  end
414
415
416  initial
417   begin
418     button_in_a = 4'b0000;
419     button_in_b = 4'b0001;
420     button_in_c = 4'b0010;
421     button_in_d = 4'b0011;
422     button_in_e = 4'b0100;
423     button_in_f = 4'b0101;
424     button_in_g = 4'b0110;
425     button_in_h = 4'b0111;
426     button_in_i = 4'b1000;
427
428     button_out_a = 4'b0000;
429     button_out_b = 4'b0001;
430     button_out_c = 4'b0010;
431     button_out_d = 4'b0011;
432     button_out_e = 4'b0100;
433     button_out_f = 4'b0101;
434     button_out_g = 4'b0110;
435     button_out_h = 4'b0111;
436     button_out_i = 4'b1000;
437
438     cik = 4'b0000;
439
440     reti = 4'b0000;
441
442  end
443
444
445  initial
446   begin
447     button_in_a = 4'b0000;
448     button_in_b = 4'b0001;
449     button_in_c = 4'b0010;
450     button_in_d = 4'b0011;
451     button_in_e = 4'b0100;
452     button_in_f = 4'b0101;
453     button_in_g = 4'b0110;
454     button_in_h = 4'b0111;
455     button_in_i = 4'b1000;
456
457     button_out_a = 4'b0000;
458     button_out_b = 4'b0001;
459     button_out_c = 4'b0010;
460     button_out_d = 4'b0011;
461     button_out_e = 4'b0100;
462     button_out_f = 4'b0101;
463     button_out_g = 4'b0110;
464     button_out_h = 4'b0111;
465     button_out_i = 4'b1000;
466
467     cik = 4'b0000;
468
469     reti = 4'b0000;
470
471  end
472
473
474  initial
475   begin
476     button_in_a = 4'b0000;
477     button_in_b = 4'b0001;
478     button_in_c = 4'b0010;
479     button_in_d = 4'b0011;
480     button_in_e = 4'b0100;
481     button_in_f = 4'b0101;
482     button_in_g = 4'b0110;
483     button_in_h = 4'b0111;
484     button_in_i = 4'b1000;
485
486     button_out_a = 4'b0000;
487     button_out_b = 4'b0001;
488     button_out_c = 4'b0010;
489     button_out_d = 4'b0011;
490     button_out_e = 4'b0100;
491     button_out_f = 4'b0101;
492     button_out_g = 4'b0110;
493     button_out_h = 4'b0111;
494     button_out_i = 4'b1000;
495
496     cik = 4'b0000;
497
498     reti = 4'b0000;
499
500  end
501
502
503  initial
504   begin
505     button_in_a = 4'b0000;
506     button_in_b = 4'b0001;
507     button_in_c = 4'b0010;
508     button_in_d = 4'b0011;
509     button_in_e = 4'b0100;
510     button_in_f = 4'b0101;
511     button_in_g = 4'b0110;
512     button_in_h = 4'b0111;
513     button_in_i = 4'b1000;
514
515     button_out_a = 4'b0000;
516     button_out_b = 4'b0001;
517     button_out_c = 4'b0010;
518     button_out_d = 4'b0011;
519     button_out_e = 4'b0100;
520     button_out_f = 4'b0101;
521     button_out_g = 4'b0110;
522     button_out_h = 4'b0111;
523     button_out_i = 4'b1000;
524
525     cik = 4'b0000;
526
527     reti = 4'b0000;
528
529  end
530
531
532  initial
533   begin
534     button_in_a = 4'b0000;
535     button_in_b = 4'b0001;
536     button_in_c = 4'b0010;
537     button_in_d = 4'b0011;
538     button_in_e = 4'b0100;
539     button_in_f = 4'b0101;
540     button_in_g = 4'b0110;
541     button_in_h = 4'b0111;
542     button_in_i = 4'b1000;
543
544     button_out_a = 4'b0000;
545     button_out_b = 4'b0001;
546     button_out_c = 4'b0010;
547     button_out_d = 4'b0011;
548     button_out_e = 4'b0100;
549     button_out_f = 4'b0101;
550     button_out_g = 4'b0110;
551     button_out_h = 4'b0111;
552     button_out_i = 4'b1000;
553
554     cik = 4'b0000;
555
556     reti = 4'b0000;
557
558  end
559
560
561  initial
562   begin
563     button_in_a = 4'b0000;
564     button_in_b = 4'b0001;
565     button_in_c = 4'b0010;
566     button_in_d = 4'b0011;
567     button_in_e = 4'b0100;
568     button_in_f = 4'b0101;
569     button_in_g = 4'b0110;
570     button_in_h = 4'b0111;
571     button_in_i = 4'b1000;
572
573     button_out_a = 4'b0000;
574     button_out_b = 4'b0001;
575     button_out_c = 4'b0010;
576     button_out_d = 4'b0011;
577     button_out_e = 4'b0100;
578     button_out_f = 4'b0101;
579     button_out_g = 4'b0110;
580     button_out_h = 4'b0111;
581     button_out_i = 4'b1000;
582
583     cik = 4'b0000;
584
585     reti = 4'b0000;
586
587  end
588
589
590  initial
591   begin
592     button_in_a = 4'b0000;
593     button_in_b = 4'b0001;
594     button_in_c = 4'b0010;
595     button_in_d = 4'b0011;
596     button_in_e = 4'b0100;
597     button_in_f = 4'b0101;
598     button_in_g = 4'b0110;
599     button_in_h = 4'b0111;
600     button_in_i = 4'b1000;
601
602     button_out_a = 4'b0000;
603     button_out_b = 4'b0001;
604     button_out_c = 4'b0010;
605     button_out_d = 4'b0011;
606     button_out_e = 4'b0100;
607     button_out_f = 4'b0101;
608     button_out_g = 4'b0110;
609     button_out_h = 4'b0111;
610     button_out_i = 4'b1000;
611
612     cik = 4'b0000;
613
614     reti = 4'b0000;
615
616  end
617
618
619  initial
620   begin
621     button_in_a = 4'b0000;
622     button_in_b = 4'b0001;
623     button_in_c = 4'b0010;
624     button_in_d = 4'b0011;
625     button_in_e = 4'b0100;
626     button_in_f = 4'b0101;
627     button_in_g = 4'b0110;
628     button_in_h = 4'b0111;
629     button_in_i = 4'b1000;
630
631     button_out_a = 4'b0000;
632     button_out_b = 4'b0001;
633     button_out_c = 4'b0010;
634     button_out_d = 4'b0011;
635     button_out_e = 4'b0100;
636     button_out_f = 4'b0101;
637     button_out_g = 4'b0110;
638     button_out_h = 4'b0111;
639     button_out_i = 4'b1000;
640
641     cik = 4'b0000;
642
643     reti = 4'b0000;
644
645  end
646
647
648  initial
649   begin
650     button_in_a = 4'b0000;
651     button_in_b = 4'b0001;
652     button_in_c = 4'b0010;
653     button_in_d = 4'b0011;
654     button_in_e = 4'b0100;
655     button_in_f = 4'b0101;
656     button_in_g = 4'b0110;
657     button_in_h = 4'b0111;
658     button_in_i = 4'b1000;
659
660     button_out_a = 4'b0000;
661     button_out_b = 4'b0001;
662     button_out_c = 4'b0010;
663     button_out_d = 4'b0011;
664     button_out_e = 4'b0100;
665     button_out_f = 4'b0101;
666     button_out_g = 4'b0110;
667     button_out_h = 4'b0111;
668     button_out_i = 4'b1000;
669
670     cik = 4'b0000;
671
672     reti = 4'b0000;
673
674  end
675
676
677  initial
678   begin
679     button_in_a = 4'b0000;
680     button_in_b = 4'b0001;
681     button_in_c = 4'b0010;
682     button_in_d = 4'b0011;
683     button_in_e = 4'b0100;
684     button_in_f = 4'b0101;
685     button_in_g = 4'b0110;
686     button_in_h = 4'b0111;
687     button_in_i = 4'b1000;
688
689     button_out_a = 4'b0000;
690     button_out_b = 4'b0001;
691     button_out_c = 4'b0010;
692     button_out_d = 4'b0011;
693     button_out_e = 4'b0100;
694     button_out_f = 4'b0101;
695     button_out_g = 4'b0110;
696     button_out_h = 4'b0111;
697     button_out_i = 4'b1000;
698
699     cik = 4'b0000;
700
701     reti = 4'b0000;
702
703  end
704
705
706  initial
707   begin
708     button_in_a = 4'b0000;
709     button_in_b = 4'b0001;
710     button_in_c = 4'b0010;
711     button_in_d = 4'b0011;
712     button_in_e = 4'b0100;
713     button_in_f = 4'b0101;
714     button_in_g = 4'b0110;
715     button_in_h = 4'b0111;
716     button_in_i = 4'b1000;
717
718     button_out_a = 4'b0000;
719     button_out_b = 4'b0001;
720     button_out_c = 4'b0010;
721     button_out_d = 4'b0011;
722     button_out_e = 4'b0100;
723     button_out_f = 4'b0101;
724     button_out_g = 4'b0110;
725     button_out_h = 4'b0111;
726     button_out_i = 4'b1000;
727
728     cik = 4'b0000;
729
730     reti = 4'b0000;
731
732  end
733
734
735  initial
736   begin
737     button_in_a = 4'b0000;
738     button_in_b = 4'b0001;
739     button_in_c = 4'b0010;
740     button_in_d = 4'b0011;
741     button_in_e = 4'b0100;
742     button_in_f = 4'b0101;
743     button_in_g = 4'b0110;
744     button_in_h = 4'b0111;
745     button_in_i = 4'b1000;
746
747     button_out_a = 4'b0000;
748     button_out_b = 4'b0001;
749     button_out_c = 4'b0010;
750     button_out_d = 4'b0011;
751     button_out_e = 4'b0100;
752     button_out_f = 4'b0101;
753     button_out_g = 4'b0110;
754     button_out_h = 4'b0111;
755     button_out_i = 4'b1000;
756
757     cik = 4'b0000;
758
759     reti = 4'b0000;
760
761  end
762
763
764  initial
765   begin
766     button_in_a = 4'b0000;
767     button_in_b = 4'b0001;
768     button_in_c = 4'b0010;
769     button_in_d = 4'b0011;
770     button_in_e = 4'b0100;
771     button_in_f = 4'b0101;
772     button_in_g = 4'b0110;
773     button_in_h = 4'b0111;
774     button_in_i = 4'b1000;
775
776     button_out_a = 4'b0000;
777     button_out_b = 4'b0001;
778     button_out_c = 4'b0010;
779     button_out_d = 4'b0011;
780     button_out_e = 4'b0100;
781     button_out_f = 4'b0101;
782     button_out_g = 4'b0110;
783     button_out_h = 4'b0111;
784     button_out_i = 4'b1000;
785
786     cik = 4'b0000;
787
788     reti = 4'b0000;
789
790  end
791
792
793  initial
794   begin
795     button_in_a = 4'b0000;
796     button_in_b = 4'b0001;
797     button_in_c = 4'b0010;
798     button_in_d = 4'b0011;
799     button_in_e = 4'b0100;
800     button_in_f = 4'b0101;
801     button_in_g = 4'b0110;
802     button_in_h = 4'b0111;
803     button_in_i = 4'b1000;
804
805     button_out_a = 4'b0000;
806     button_out_b = 4'b0001;
807     button_out_c = 4'b0010;
808     button_out_d = 4'b0011;
809     button_out_e = 4'b0100;
810     button_out_f = 4'b0101;
811     button_out_g = 4'b0110;
812     button_out_h = 4'b0111;
813     button_out_i = 4'b1000;
814
815     cik = 4'b0000;
816
817     reti = 4'b0000;
818
819  end
820
821
822  initial
823   begin
824     button_in_a = 4'b0000;
825     button_in_b = 4'b0001;
826     button_in_c = 4'b0010;
827     button_in_d = 4'b0011;
828     button_in_e = 4'b0100;
829     button_in_f = 4'b0101;
830     button_in_g = 4'b0110;
831     button_in_h = 4'b0111;
832     button_in_i = 4'b1000;
833
834     button_out_a = 4'b0000;
835     button_out_b = 4'b0001;
836     button_out_c = 4'b0010;
837     button_out_d = 4'b0011;
838     button_out_e = 4'b0100;
839     button_out_f = 4'b0101;
840     button_out_g = 4'b0110;
841     button_out_h = 4'b0111;
842     button_out_i = 4'b1000;
843
844     cik = 4'b0000;
845
846     reti = 4'b0000;
847
848  end
849
850
851  initial
852   begin
853     button_in_a = 4'b0000;
854     button_in_b = 4'b0001;
855     button_in_c = 4'b0010;
856     button_in_d = 4'b0011;
857     button_in_e = 4'b0100;
858     button_in_f = 4'b0101;
859     button_in_g = 4'b0110;
860     button_in_h = 4'b0111;
861     button_in_i = 4'b1000;
862
863     button_out_a = 4'b0000;
864     button_out_b = 4'b0001;
865     button_out_c = 4'b0010;
866     button_out_d = 4'b0011;
867     button_out_e = 4'b0100;
868     button_out_f = 4'b0101;
869     button_out_g = 4'b0110;
870     button_out_h = 4'b0111;
871     button_out_i = 4'b1000;
872
873     cik = 4'b0000;
874
875     reti = 4'b0000;
876
877  end
878
879
880  initial
881   begin
882     button_in_a = 4'b0000;
883     button_in_b = 4'b0001;
884     button_in_c = 4'b0010;
885     button_in_d = 4'b0011;
886     button_in_e = 4'b0100;
887     button_in_f = 4'b0101;
888     button_in_g = 4'b0110;
889     button_in_h = 4'b0111;
890     button_in_i = 4'b1000;
891
892     button_out_a = 4'b0000;
893     button_out_b = 4'b0001;
894     button_out_c = 4'b0010;
895     button_out_d = 4'b0011;
896     button_out_e = 4'b0100;
897     button_out_f = 4'b0101;
898     button_out_g = 4'b0110;
899     button_out_h = 4'b0111;
900     button_out_i = 4'b1000;
901
902     cik = 4'b0000;
903
904     reti = 4'b0000;
905
906  end
907
908
909  initial
910   begin
911     button_in_a = 4'b0000;
912     button_in_b = 4'b0001;
913     button_in_c = 4'b0010;
914     button_in_d = 4'b0011;
915     button_in_e = 4'b0100;
916     button_in_f = 4'b0101;
917     button_in_g = 4'b0110;
918     button_in_h = 4'b0111;
919     button_in_i = 4'b1000;
920
921     button_out_a = 4'b0000;
922     button_out_b = 4'b0001;
923     button_out_c = 4'b0010;
924     button_out_d = 4'b0011;
925     button_out_e = 4'b0100;
926     button_out_f = 4'b0101;
927     button_out_g = 4'b0110;
928     button_out_h = 4'b0111;
929     button_out_i = 4'b1000;
930
931     cik = 4'b0000;
932
93
```

```

Ln# 1 module adder(player1_in, player2_in, sum, wongame, didnotwingame);
2   input [0:3] player1_in;
3   input [0:3] player2_in;
4   output [0:3] sum;
5   output wongame;
6   output didnotwingame;
7
8   reg [0:3] sum;
9   reg wongame;
10  reg didnotwingame;
11
12  always@(player1_in, player2_in)
13    begin
14      sum = player1_in + player2_in;
15      case(sum)
16        4'b1111:
17          begin
18            wongame = 1;
19            didnotwingame = 0;
20          end
21        default:
22          begin
23            wongame = 0;
24            didnotwingame = 1;
25          end
26      endcase
27    end
28  endmodule
29

```

Figure 73: Adder module Verilog code screen capture

```

Ln# 1 `timescale 1ns/100ps
2 module Testbench_adder();
3   reg [0:3] player1_in_s;
4   reg [0:3] player2_in_s;
5   wire [0:3] sum_s;
6   wire wongame_s;
7   wire didnotwingame_s;
8
9
10  adder adder_DUT(player1_in_s, player2_in_s, sum_s, wongame_s, didnotwingame_s);
11  initial
12    begin
13      player1_in_s = 4'b1101; player2_in_s = 4'b0010;
14      #10 player1_in_s = 4'b0000; player2_in_s = 4'b0010;
15      #10 player1_in_s = 4'b0000; player2_in_s = 4'b0000;
16      #10 player1_in_s = 4'b1111; player2_in_s = 4'b0010;
17      #10 player1_in_s = 4'b1001; player2_in_s = 4'b0010;
18    end
19  endmodule
20

```

Figure 74: Adder test bench module Verilog code screen capture