Memory Tester Game

Table of Contents

**Introduction1**

**Description1**

**Black Box1**

ROM User ID Module2

ROM Password Module2

RAM Password Reset Module2

**Design4**

ROM User ID Module5

ROM Password Module5

RAM Password Reset Module5

Level Controller Module5

Game Module5

Timer Module5

Score Table Module5

**Simulations4**

**Results4**

**Conclusion4**

**INTRODUCTION:**

Minds Mapped is a multi-player game which has been developed using FPGA board DE2-115. The idea behind ‘brain training’ is that if you practice a task that taps a core component of cognitive ability, like memory, the training will improve your ability to perform other tasks that also rely on memory, not just in the lab, but also in the world. That premise is known as ‘transfer-of-training’. So with this game we tried to contribute an initiative towards brain training with utilizing all the concepts that we exercised during our LABs and some real life modules.

In this game, the player has to memorize the sequence of flashing random numbers from the seven segment display and enter those numbers in the same order using the toggle switches assigned for the User Input. After all the numbers have been flashed upon the seven segment display, the countdown timer starts within which the player has to enter their input. The game iterates over a maximum of 5 levels, each level with varying values of the length of the sequence displayed and the countdown timer. If the player matches the entire sequence, he wins the round and goes on the next level. The scores of individual players will be stored along with the highest score recorded.

The player who gets the highest score will be declared with a separate green LED being turned on after the player has logged in. The winner of the game is declared by looking up at this green LED.

**DESCRIPTION:**

Before the game begins, the players are provided with their User ID and Password. Only one player can play the game at a time. The player has to first authenticate the game using his/her 16-bit User ID and password through the designated toggle switches and the overload button to start the game. Our game comes with a total of pre-defined User ID’s and passwords for our team members and also an additional Guest User ID and password. The player also has a choice to reset their password after they login initially. But once the player starts playing the game, password resetting is not possible. Password resetting has to be one immediately after the player los in. After successful authentication, the player is all set to play the game.

Once the player has logged in, he/she has to push the RNG button to generate the sequence of flashing random numbers. After all the numbers have been displayed, the countdown timer starts and the player then has to enter the binary forms of the numbers using the assigned toggle switches. If the player has entered the entire sequence correctly, he jumps to the next level. The player once again has to push the RNG button for the next level. Meanwhile, the individual score of the player will be stored along with the highest score recorded. A single toggle switch is used to see the scores of the players and the highest score. If the toggle switch is set to one, the individual score is displayed and if it is set to 0, the highest score will be displayed.

The player can continue to play the game even if he/she fails is any of the levels by pushing the RNG button again but if a failed case is encountered, the individual score will be driven back to zero which means the scores from previous levels no longer count. Also, a player can logout of the game anytime and log back in to continue to play the game from the same level where he left the game.

The overall game architecture is subdivided into several other nested modules such as: Authentication module, Level table module, Timer module, Game module, Score table module etc. All these sub modules have been explained further.

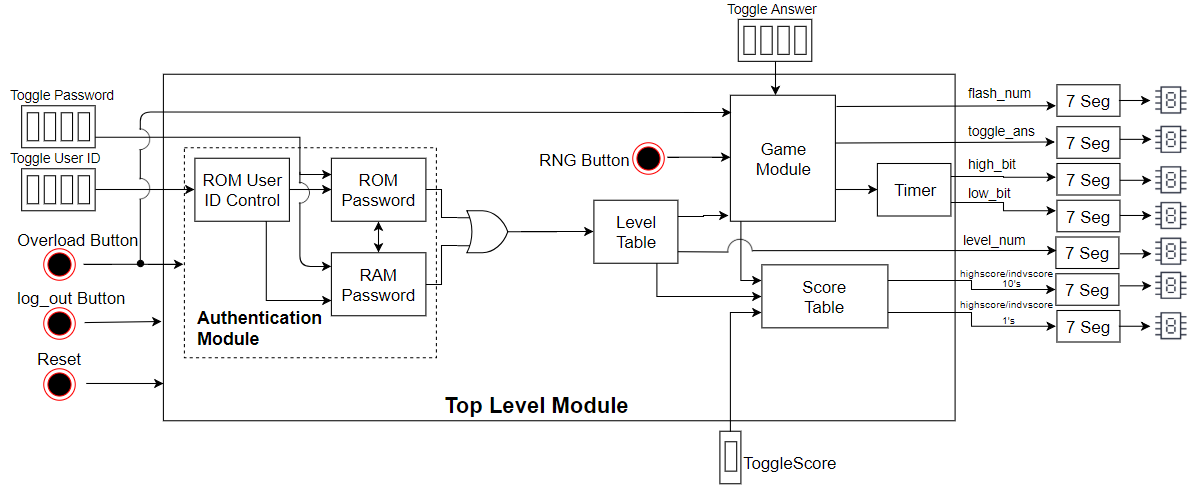
**Overload button:**

This button is specially being mentioned because, this acts as a multipurpose button have too many functionalities aligned with it.

1. It acts as authentication begin button
2. Acts as userid, password load button
3. After authentication, it acts as ‘password reset request’ button
4. Again reset password loading button
5. Once game begins, it acts as answer load button
6. After loading all numbers, it acts as confirmation button to re-confirm answer. Only then win/loose decision is made.

Each of these functionalities are encountered further.

**ARCHITECTURE:**

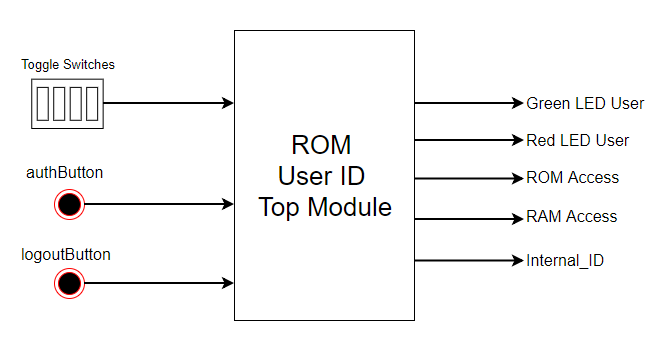


The above figure shows the overall architecture of our game. We can observe that the Authentication module consists of several submodules namely ROM User ID Control, ROM Password, and RAM Password Reset. These three modules together work for the successful authentication of the game and shows the output through Red and Green LED’s. After that this module sends the obtained output to the Level table module which displays which level the player is in on to the seven segment display. Based on the outputs from the Level table module, the Game module and the Score Table modules will be initiated. The Game module is nothing but the random sequence generator module which generates a sequence of random numbers and displays them on to the seven segment display with a time gap of 1 second for every digit along with a comparision module to make win/loose decision. The score table module keeps track of the individual scores of the players based on the game module output and also the highest score recorded among all the players.

**BLACK BOX:**

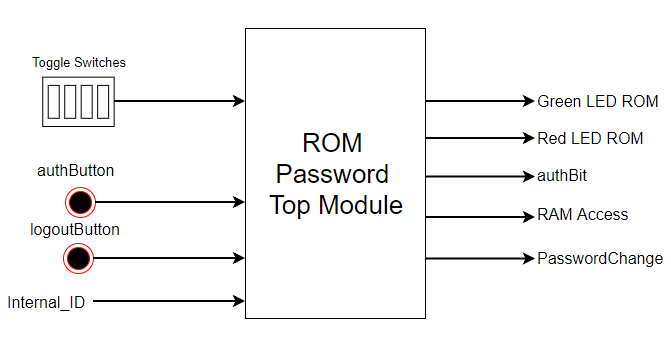
1. **ROM User ID Module:**

The ROM User ID module takes in the inputs from the assigned toggle switches (4) and push buttons namely the overload button (which operates to begin the authentication and also to load the User ID) and logout button and gives 1-bit outputs to the red and green LED’s to indicate the status of User ID input and other outputs namely ROM Access, RAM Access and Internal\_ID. The ROM and RAM Access signals along with the Internal\_ID signal trigger either the ROM Password module to check with the stored password (or) the RAM Password Reset module to allow the player to reset the password or use the already reset password for future logins. The logout button enables a player to logout from the game anytime and reset all of it to zero to allowing another player to play the game but the status of the player (like his level, score and password changes are all stored for future logins to resume back to the same status).

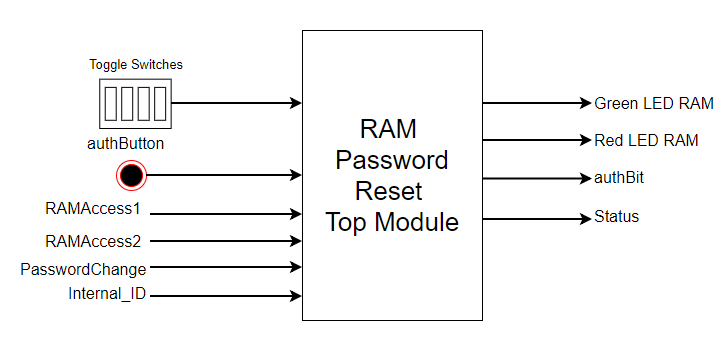
****

1. **ROM Password Module:**

This module takes in inputs from the assigned toggle switches (4), the overload button (which acts as a password load button), the logout button and Internal\_ID and gives outputs through red and green LED’s which will indicate the status of authentication. Internal ID generated from ROM USER ID Module is fed to this module to map to the corresponding password. Enetered 16 bit password is verified against the password read from this internal ID address. When authenticated green led is set to ‘1’ and when wrong red led is set to ‘1’.The output RAM access is set to ‘1’ whenever the player requests to reset password by pressing the overload button immediately after he logs in . This RAM access activates the RAM Password reset module and Passwordchange signal is sent to RAM Password reset Module to indicate it, that it is activated for password resetting and not reading password. authBit output will be set to ‘1’ when correct used id and password are given and is used to trigger all the other modules required to play the game.



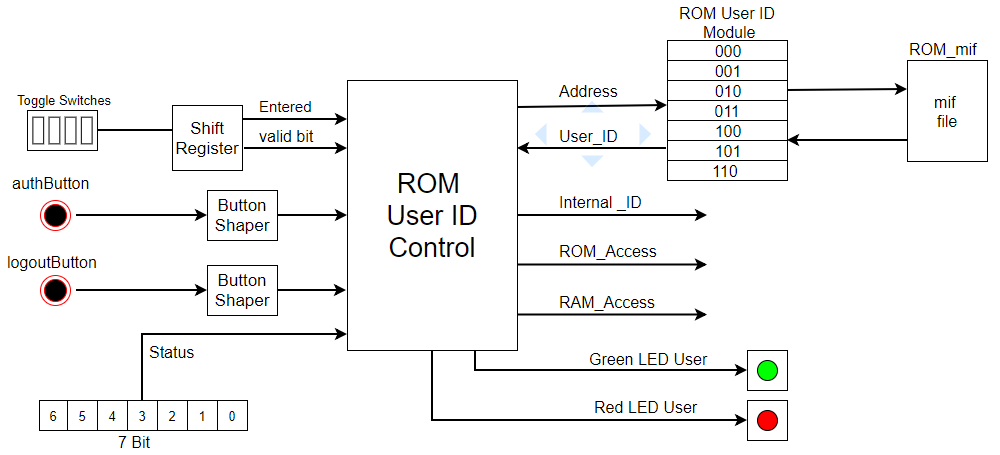
1. **RAM Password Reset Module:**



RAM Password Reset Module has two different functionalities. One is to reset password and the second is, if the player already changed his password, all his next logins are authenticated through this module. Just like the other authentication submodules, this module also takes input from toggle switches following every auth(overload) button push. It is activated upon receiving either of the access signals namely RAMAccess1 or RAMAccess2. The access functionality is decided based upon ‘PasswordChange’ signal. If it is set to zero and RAMAccess1( from USER ID Module) is set to ‘1’, that means this module is activated for checking password entry and authentication. But if Passwordchange is set to ‘1’ and RAMAccess2 is set to ‘1’ that means, the module is activated for password resetting and hence starts writing toggle entries after every overload button push to the 16 bit password at the location corresponding to internal\_ID. When password is changed green led is turned ON and also status register(7 bit) is updated to 1 at an index corresponding to internal\_ID. Like if internal\_ID is 3’b010, then status register status[6:0] is updated at status[2] to 1. So now the status register values is 7’b0000010. This value is fed to ROM USER ID Module to monitor future logins and redirect access to RAM , if password is reset.

**DESIGN:**

1. **ROM User ID Module:**

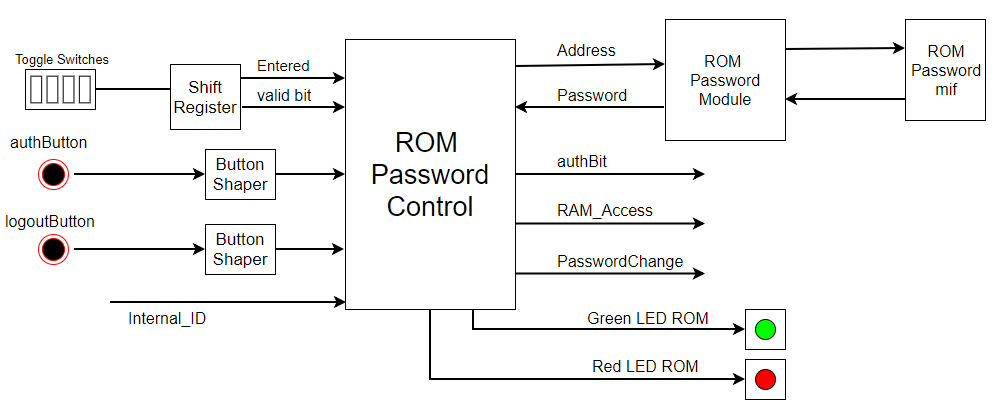


Player has to press auth button to begin authentication process.

Then the player has to enter his 16 bit user ID using toggle switches (4) and then load each 4 bit entry using an auth(overload) button. Each of these entries are fed to shift register and shifted to the left, four at a time. Auth button is wired in such a way so that it acts as shift left button to the shift register. Once all 16 bits are loaded in the shift register, the 16 bit entered value and valid bit indicating that 16 bit entry is done, is given to the ROM User ID control.

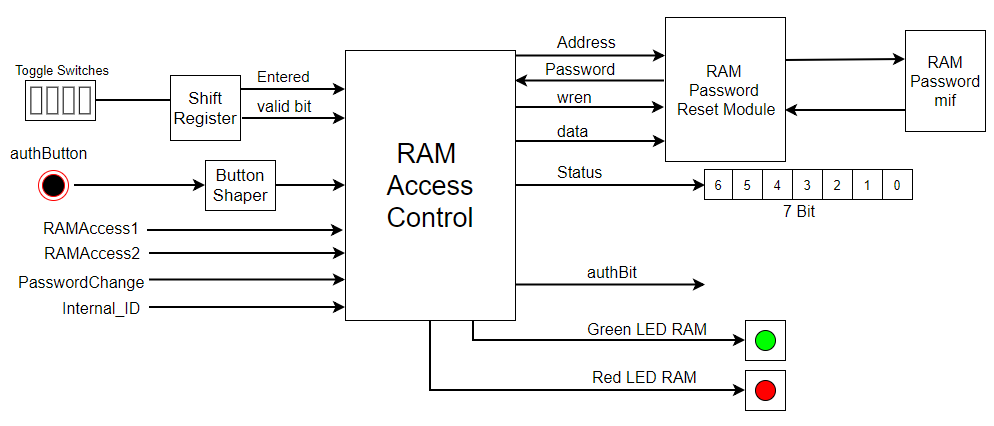
Then the, module starts to compare the entered user id with all existing User IDs in the ROM storage. If it matches with any, that location is taken as an internal ID and green led is turned ON. Else, a red led is turned ON. Once this is done, the controller checks the status register if this particular user has made any password resets previously. If status[internal\_ID] is ‘1’ RAM access is given else ROM access is given.

1. **ROM Password Module:**



Similar to Rom User id control, inputs are loaded to shift register using toggle switches and auth button. With the internal\_ID received, the control moves to a location corresponding to internal\_ID and checks toggle entries. If it matches green led is turned ON and auth\_bit is set to ‘1’ , else red led is ON. If user requests to reset password by pressing the same auth button (overload- now acts as ‘begin reset’ button) after successful authentication, then RAM access is given with passwordchange signal set to ‘1’

1. **RAM Password Reset Module:**



This module is either activated by RAM access 1 received from ROM User ID module or ROM Password module. RAM Password Reset Module has two different functionalities. One is to reset password and the second is, if the player already changed his password, all his next logins are authenticated through this module. Just like the other authentication submodules, this module also takes input from toggle switches following every auth(overload) button push. It is activated upon receiving either of the access signals namely RAMAccess1 or RAMAccess2. The access functionality is decided based upon ‘PasswordChange’ signal. If it is set to zero and RAMAccess1( from USER ID Module) is set to ‘1’, that means this module is activated for checking password entry and authentication. But if Passwordchange is set to ‘1’ and RAMAccess2 is set to ‘1’ that means, the module is activated for password resetting and hence starts writing toggle entries after every overload button push to the 16 bit password at the location inside RAM corresponding to internal\_ID. When password is changed green led is turned ON and also status register(7 bit) is updated to 1 at an index corresponding to internal\_ID. Like if internal\_ID is 3’b010, then status register status[6:0] is updated at status[2] to 1. So now the status register values is 7’b0000010. This value is fed to ROM USER ID Module to monitor future logins and redirect access to RAM , if password is reset. For future logins, it acts as a general password check.

1. **Level Controller Module:**

The entire level controller module is dependent upon I/O signals from access control (authentication), game module. State machine inside the controller manages the read/write operation with the RAM TABLE (mif file). Level check, increment, and update is done based on user authentication, player win, and input pulse. 4-bit level number is passed on to seven segment decoder for display.

A screenshot of a cell phone

Description generated with very high confidence

I/O Description:

|  |  |  |
| --- | --- | --- |
| Signal Name | Description | Data /Size |
| Internal\_ID | Unique user ID coming from authentication module. | [2:0] 3 bits |
| authBit | Authorization bit coming from authentication module indicating a successful authentication process. | Boolean |
| rng\_buttton | Provides a trigger pulse. | Boolean |
| logoutBUttton | Indicates if the user is still logged in and playing | Boolean |
| Green LED User | Indicates successful authentication attempt. | Boolean |
| Win | Indicates a player’s win at a specific level | Boolean |
| Addresss | Internal address to identify a RAM slot for R/W operation | [2:0] 3 bits |
| level\_o | Outgoing level data for RAM table. | [2:0] 3 bits |
| Wren | Write enable signal (it must be 1 to write, if 0 it will only read given 2 delay cycles). | Boolean |
| level\_i | Incoming level data for RAM table. | [2:0] 3 bits |
| Level\_num | Indicates the player’s level. Outgoing signal to 7 seg decoder for user display. Also input to game module and score table module. | [3:0] 4 bits |
| Level\_Updated | Outgoing signal in to the game module if the level has been updated. | Boolean |

Table 1:I/O signal description for Level Controller and Table

## FSM Level Controller:

A close up of a map

Description generated with high confidence

**RAM Level Table:**

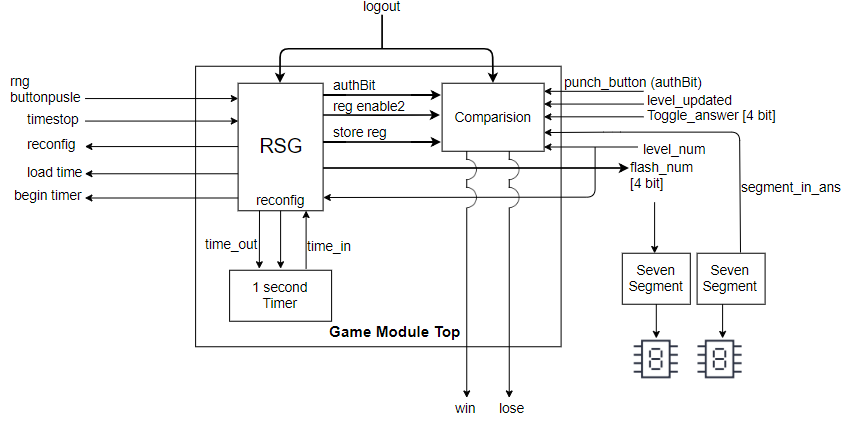
This data is stored and associated with a memory initialization file (mif). Data can be read from this table, but can only be written with input signal ‘wren’ set to high. It is important for to wait for 2 clock cycles for data read or write.

|  |  |  |
| --- | --- | --- |
| Game Level | RAM Address | Data [2:0] |
| 1 | 001 | XXX |
| 2 | 010 | XXX |
| 3 | 011 | XXX |
| 4 | 100 | XXX |
| 5 | 101 | XXX |

Table 2: shows the setup inside the RAM level table

1. **Game Module:**

Game module consists of further sub modules namely RSG, Comparison and a 1 second timer. All these sub modules together contribute in generating the sequence of flashing random numbers onto the seven segment display for every button push.

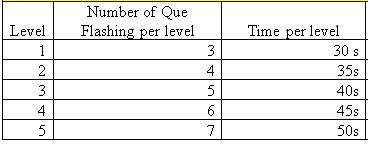


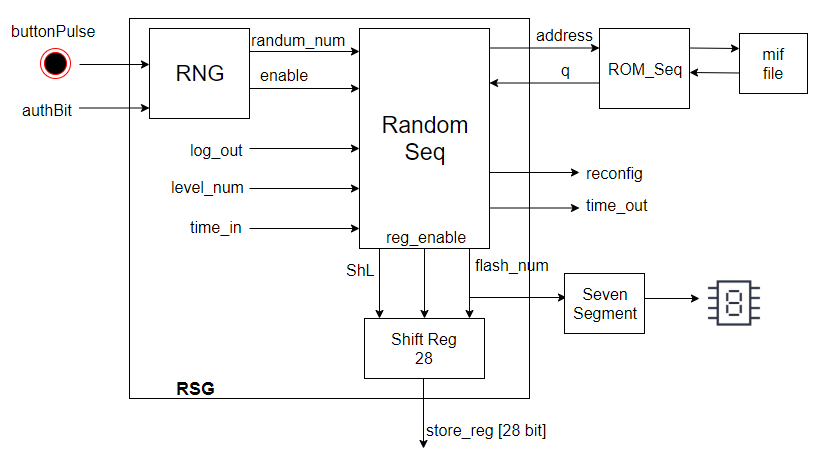
**RSG (Random Sequence Generator):**

The random sequence generator module further consists of more sub modules namely RNG (Random Number Generator), Random Seq and Shift Register and 1 second timer instantiation. Once player presses the rng\_button, random number is generated by RNG module. With the random number an enable signal is sent to the Random\_seq module. Random logic is applied to get random addresses according to a count value set by the module for each level. For level 1 count value is set to ‘3’ , for ‘2’ it is ‘4’ and so on until level ‘5’ it is ‘7’. So that many random addresses are generated and from these addresses randomly placed numbers are extracted. Flash\_num output variable is updated each time with these extracted random numbers and holds the value for one second and no number for next 1 second alternately to give a number flashing effect on board, where numbers are blinked not just displayed. For this, a timeout signal is sent for each flash\_num update, after 1 second time\_in signal is returned from 1 second timer and then empty number is flashed the same way for 1 second, then moves to a new update value. Simultaneously count value which is set for each level is decremented and monitored. When reaches ‘0’, a global reconfigurable timer that acts as a count down timer is instantiated with time values varying for each level. These varying time values and count down scale are given using input\_num and reconfig values to the timer respectively. Refer to the table below. When numbers are flashed the numbers are automatically stored in shift register. For this storage is enabled only when numbers start flashing, when count goes to ‘0’ storage is disabled.

As soon as win or loose signal is received timer is stopped and reset back to initial values by sending signals from inside the Random\_seq module itself.

Also, when all numbers are flashed, comparision module storage is enable by sending signal from this module indicating “number flashing is done, user input can now be taken”. Also the already stored flash\_num values are also fed to comparision module, for future comparision.

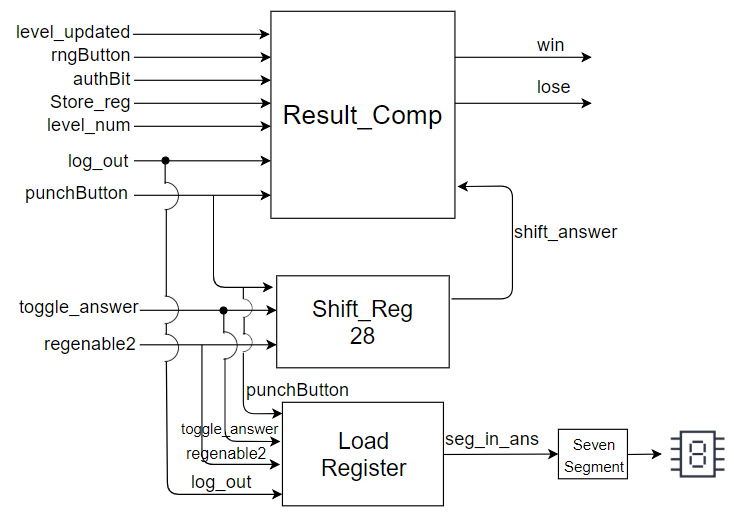




**Comparison:**

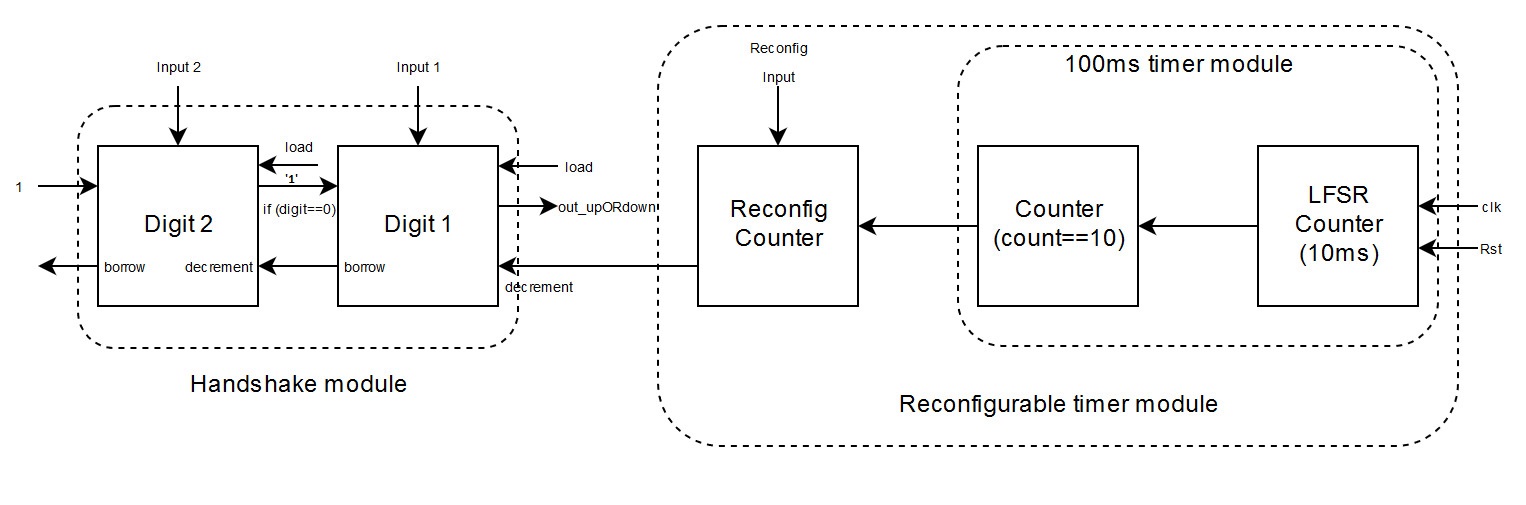
Once it is enabled by reg\_enable2 signal from RSG, comparison module is ready to receive input from user. The user inputs are stored in a shift register and then fed to result\_comp module.The result\_comp module compares the two storage registers to see if there is a matching of user input and flashed numbers. A win/loose decision is made based on this. Also a time\_stop signal from global timer is monitored. When timer reaches ‘0’ before all entries are made by the user, that means count down ends, hence, that is also considered as a loose. Loose is indicated by red signal whereas win signal just drives level and score to increment.

Also user inputs are fed to load register each time punch\_button(overload) is pressed and the values are displayed. After all entries, player has to confirm his answer gain with the same confirmation\_button(overload) and only then win/loose decision is made.



1. **Timer Module:**

Timer module consists of 2 sub modules i.e. Reconfigurable Timer Module and Handshake Module.



**Reconfigurable Timer Module:**

Once user press the button, Inverted button push pulse from Button Inversion module triggers the LFSR counter. This LFSR counter module is designed to generates the pulse for every 10 milliseconds (i.e. 500,000 clock cycles). This pulse is then fed to another counter module which counts its signals and generate pulse for every incoming 10th pulse from LFSR module. Overall these two modules generate the pulse for every 100ms.

The pulse generated from counter module is then fed to the Reconfigurable counter module depending on the reconfigurable input value we can control the speed of the timer. If the reconfigurable input of this module is set to 10, it means that count value of this counter module is set to 10 and it generates a pulse for every 10th signal i.e. for every 1 sec and similarly if input is set to 5, then module generates a pulse for every half second. Basically, the input of this module decides the number of incoming signal required to generate a pulse from this (Reconfigurable) Module.

**Handshake Timer Module:**

The Pulse generated from Reconfigurable Timer module is then fed to this module.  This module has two counters (outputs) connected to the seven-segment display. Each counter represents a digit and cannot start until the previous digit reached “0”.

At start of the game these two counters are set to a value. Upon receiving signal from Reconfigurable Timer module, the counter value at one’s digit module start decrementing by 1 value until it reaches 0. By ‘Handshake technique’ whenever the digit module at ‘ones’ place reaches 0 it sends the borrow signal which act as ‘decrement’ signal to ‘tens’ place digit module. In response, tens place digit module sends stop signal whose value is either ‘0’ or ‘1’ (1 means one’s place digit counter module can’t be incremented and 0 means it can be incremented to 9). The stop signal is ‘1’ when tens place digit counter module is zero. This stop signal is then fed to increment\_upstream input of one’s place module. So, whenever one’s place module receives increment\_upstream as 1 then this counter stops incrementing to 9 from zero.

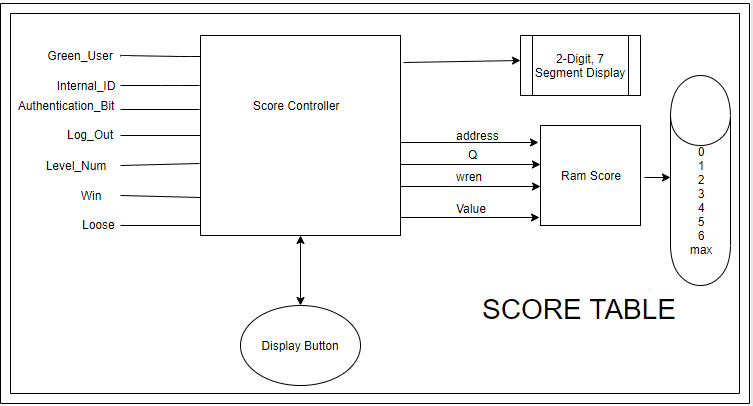
Also, whenever tens place digit counter receives borrow signal (which act as decrement signal to tens place digit counter module) from previous one’s place digit module, its counter will decrement by 1 value until it reaches zero. Once both the digit counter module reach zeros the timer module stops.

The digit counter module is independent module, so whenever we need to increase the digits in the timer we just need to instantiate one more module in the top module rather than changing whole code. Thus, this module is very scalable.

1. **Score Table Module:**

**Score Table**

**Architecture Diagram**

****

**Description:**

**Score Table** is the orchestratorwhich has internally 3 more modules within itself.

1. **Score controller**
2. **RAM module**
3. **2 digit 7 segment display**

**Score controller:** This is the main module which does all arithmetic and logical calculation for score based on various conditions .So in general this module gets triggered whenever the ever we set a enable switch that is **auth\_bit to 1** and it is triggered by the prior module called GAME MODULE . Whenever a player successfully matches the flashed numbers this module is triggered to store his score for that level. So this module is triggered for every module be it is a win or loss. For every win (i.e flash number authentication is successful at every level) score stored at RAM gets incremented by a weight (i.e in our case it is same as level number) against the player ID. So basically we have 4 RAM module operations involved here as mentioned below.

**Score Table Insights:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Score Table** | | | | |
| Level | Number of Que Flashing per level | Time per level | Level Score | Cumulative score |
| 1 | 3 | 30 s | 1 | 1 |
| 2 | 4 | 35s | 2 | 3 |
| 3 | 5 | 40s | 3 | 6 |
| 4 | 6 | 45s | 4 | 10 |
| 5 | 7 | 50s | 5 | 15 |

**RAM Read calls:**

1. Individual Player for Level score read
2. Max score read

**RAM Write calls:**

1. Individual Player for Level score update
2. Max score update

**RAM module:** This is theinterfacing code forinteracting with the hardware for accessing the RAM table .So RAM table consist of word length of 8 bits and 8 rows with each row corresponding to playerID. Here we have two operations . RAM Read and RAM write operations and they are separated from each other by a Boolean parameter called **write\_enable** to 1 for writing and 0 for reading .

|  |  |
| --- | --- |
| **RAM TABLE** | |
| ADDRESS | VALUE |
| PlayerID\_1 | 0 (initial values) |
| PlayerID\_2 | 0 |
| PlayerID\_3 | 0 |
| PlayerID\_4 | 0 |
| PlayerID\_5 | 0 |
| PlayerID\_6 | 0 |
| Guest Player | 0 |
| MAX\_Scorer | 0 |

**2 digit 7 segment display:** This is a basic score converter which has mapping it according to 7 segment decoder starting forma range of 000000 to 111111 = 64 (Max number can be displayed).

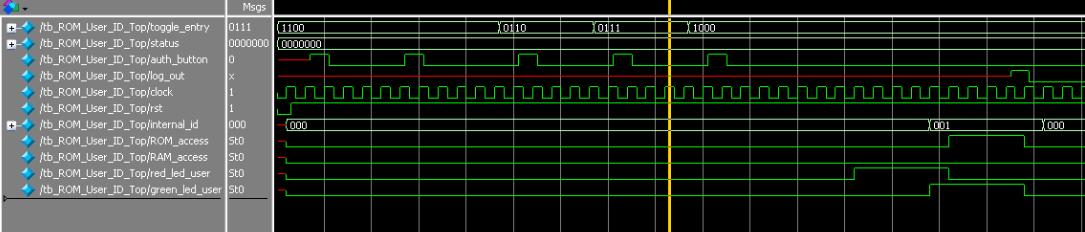
**I/O Description : Table 1:I/O signal description for score table**

|  |  |  |
| --- | --- | --- |
| Signal Name | Description | Data /Size |
| Level | Level number coming from game module. | [2:0] 3 bits |
| authBit | Authorization bit coming from authentication module indicating a successful authentication process. | Boolean |
| rng\_buttton | Provides a trigger pulse. | Boolean |
| logoutBUttton | Indicates if the user is still logged in and playing | Boolean |
| Green LED User | Indicates successful authentication attempt. | Boolean |
| Win | Indicates a player’s win at a specific level | Boolean |
| Addresss | Internal address to identify a RAM slot for R/W operation | [2:0] 3 bits |
| level\_0 | Outgoing level data for RAM table. | [2:0] 3 bits |
| Wren | Write enable signal (it must be 1 to write, if 0 it will only read given 2 delay cycles). | Boolean |
| internal\_id | Player unique Id used for RAM addressing | [2:0] 3 bits |
| disp\_button | This is connected to a toggle switch which shows individual score when 0 and max score when 1 | [3:0] 4 bits |
| green\_max | Outgoing signal denoting max score | Boolean |

**SIMULATIONS:**

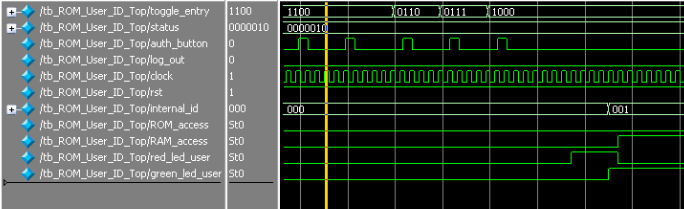
1. **ROM User ID Module:**

ROM giving no RAMaccess since the status register is all set to 0000000



16 bit user ID is given. A quick comparision is done with the userIDs available in ROM storage. When each userID is compared, if it doesn’t match, a red led is turned ON. But sequentially when compared one after the other and finally reaches the right one, red led is turned OFF and green led is turned ON. This short period for which red led is ON while comparing is not visible to human eye on the board since the comparision takes place very fast and green led remains ON and hence can be seen. But if UserID doesn’t match any stored userIDs , red led remains ON and can be seen by the user. Once the userID matches, the internal\_ID is updated with the location of that particular userID. In this case it becomes 001. By checking the status register since all the bits are set to ‘0’, particularly status[internal\_ID] is set to ‘0’, so, no RAM access is given, as no previous password changes occurred. Instead ROM access is given for password check. Once you logout, everything is set back to initial values.

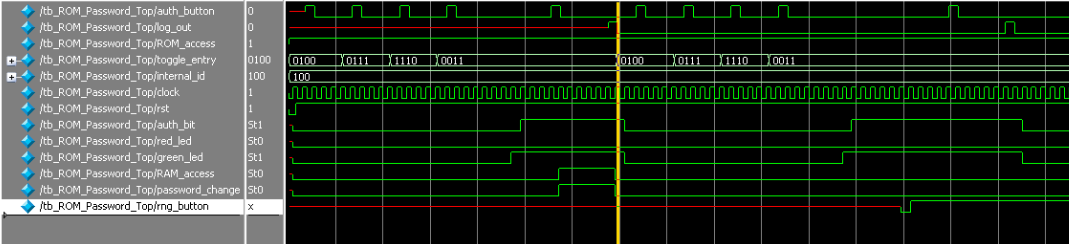
ROM User ID giving RAM access as status register is set to ‘1’ at index ‘2’:



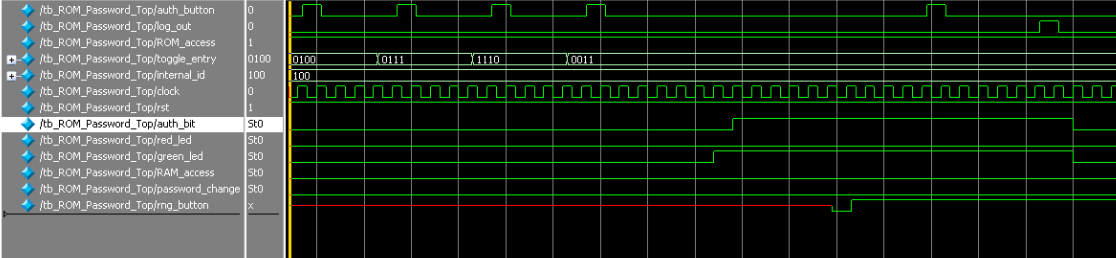
Similarly, userID is again given, since it matches, green led is turned ON and internal\_ID is set to ‘001’. But in this case, status register is set to ‘1’ at location status[internal\_ID] i.e; status[010] is a ‘1’, so previous password resets have occurred and hence, RAM access is set to ‘1’

1. **ROM Password Module:**

ROM Password Top module:

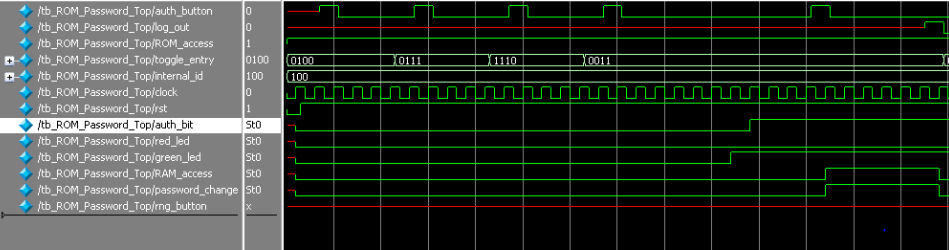


ROM Password RAM access not given for passwordchange once game begins:



Here the password is compared at particular internalID address, and when matches authbit is et to ‘1’ and green led is turned ON. After, authentication success, player has option to reset his password immediately on RAM by pressing auth button again. But after he starts playing, this option is disabled. Player has to re-login so as to reset his password. This is done by monitoring rng\_button. If rng\_button is set to ‘0’, that means player started playing, hence even if the auth button is pressed after authentication, RAM access is not given to reset password

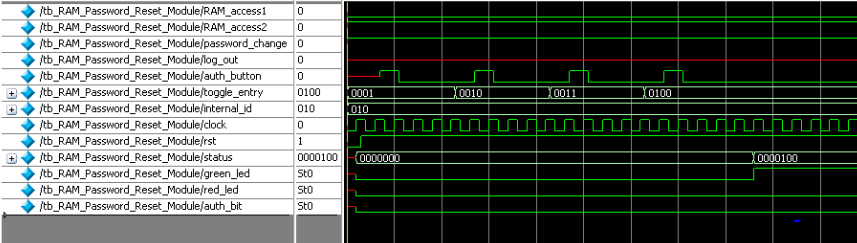
ROM Password RAM access for password reset enabled, no rng\_button pulse:



In this case, it works exactly the opposite. Here, rng\_button is not set to ‘0’,hence, when auth button is pressed again after successful authentication, RAMaccess and password change pulses are set to ‘1’.

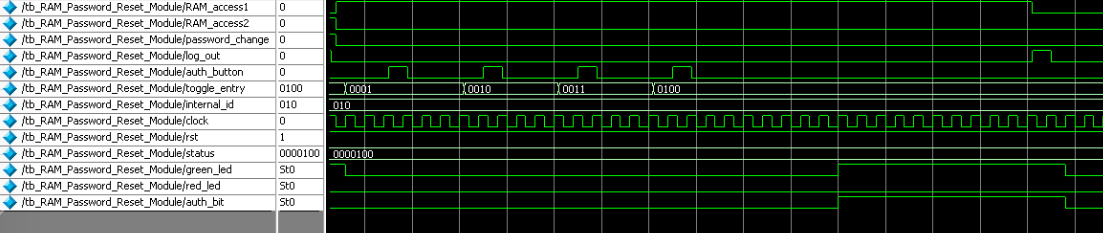
1. **RAM Password Module:**

RAM Access 2 Password Reset case:



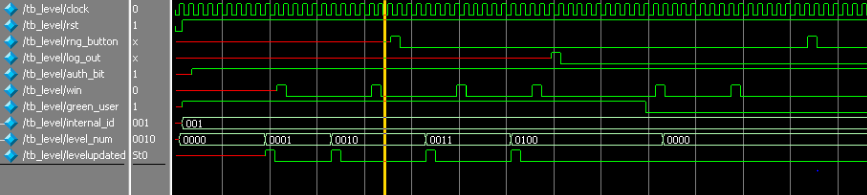
RAMaccess 2 and passwordchange are set to ‘1’, so password resetting takes place. Once password is reset, status register is updated with ‘1’ at that particular internal ID location. Green led is turned ON to indicate password resetting is successful.

RAM Access 1 Password check case:



Usual, password check takes place by comparing password at internal ID location with toggle entries. Green led is turned ON and authbit is set to ‘1’ on successful authenticatiom.

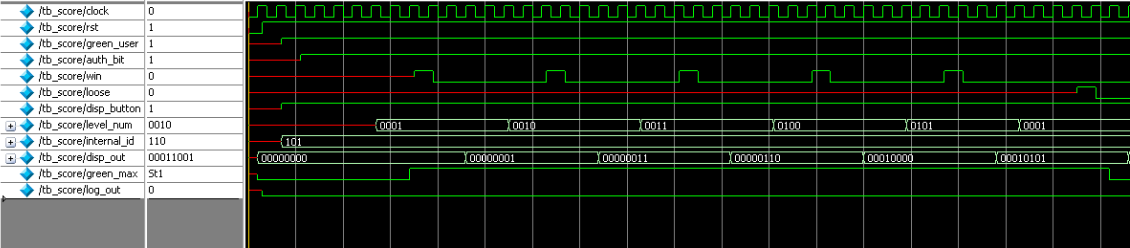
1. **Level Table Module:**



Level is retrieved as soon as userID, is verified. A for each ‘win’ signal set to ‘1’ level is updated by ‘+1’.

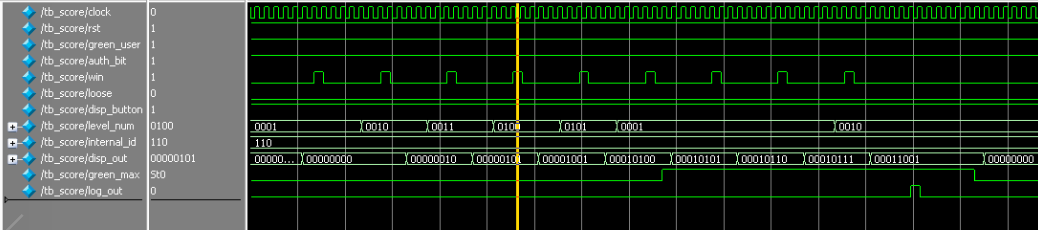
1. **Score Table Module:**

Player-1 max score:



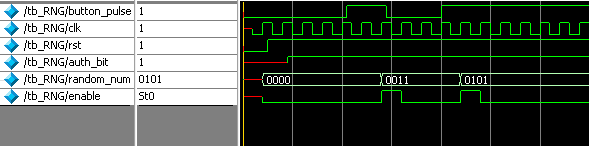
Player1, playing the game for the first time, no previous maximum were recorded, hence module assumes player1 to be maximum scorer by default. Each ‘win; signal set to ‘1’ results in a score increment. And a loose set to ‘1’ sets back score to 0 as per game rules. Disp\_button is set to ‘1’ all the time, two show the score changes. Instead of a hexadecimal conversion, the score display shows decimal scores due to a conversion module. For example, a score of 15 disp\_out is shown as 0001 0101 and not 00001111.

Player-2 max score:

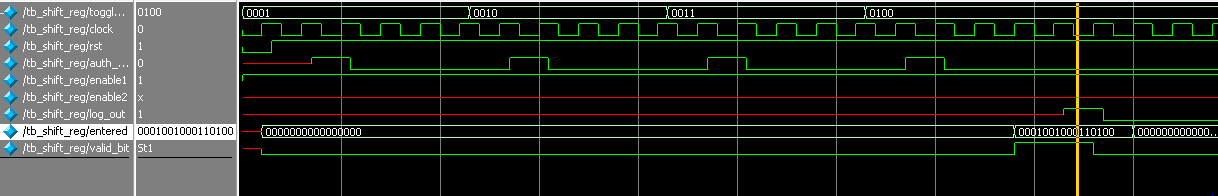


Player2 playing the game, already player1’s maximum score is recorded as team highs core, hence player2 green\_max led is OFF initially, since he is not the maximum scorer. But once, he starts playing and scores more than previous registered max score of ‘15’, green\_max led is ON. Once logged out, everything is set back to initial state.

**RNG simulations:**



**Shift register for storing flashed numbers and toggle answer:**



**RESULTS:**

A successfully functioning game module with authentication is implemented.

**Appendix:**

**//Reconfigurable counter**

module reconfig\_count(pulse,clk,rst,reconfig,outpulse);

input clk,rst,pulse;

input [3:0]reconfig;

output reg outpulse;

reg [3:0] count;

always @(posedge clk)

begin

if(rst == 0)

begin

count<=4'b0000;

outpulse=0;

end

else

begin

if(pulse==1) begin

if(count>reconfig) begin

count<=4'b0000;

end

count <=count+4'b0001;

if(count==reconfig) begin

outpulse=1;

count<=0;

end

else begin

outpulse=0;

end

end

else begin

if(outpulse==1)begin

outpulse=0;

end

end

end

end

endmodule

//result\_comp

//MISSION-V

//module for comparing entered sequence and random sequence

module result\_comp(time\_stop,levelupdated,logout,rng\_button,auth\_bit,punch\_button,shif\_answer,store\_reg,level\_num,win,loose,clock,rst);

input logout,rng\_button,punch\_button,auth\_bit,levelupdated,time\_stop;

input [27:0] shif\_answer,store\_reg;

input [3:0] level\_num;

input clock,rst;

output reg win,loose;

reg [2:0] count;

reg [2:0] state;

parameter INIT=0,wait\_for\_rng=1,comp\_res=2,wait\_for\_level=3;

always @(posedge clock)

begin

if(rst==0)

begin

count<=3'b000;

win<=0;

loose<=0;

state<=INIT;

end

else

begin

case(state)

INIT:

begin

win<=0;

loose<=0;

case(level\_num)

4'b0001:

count<=3'b011;

4'b0010:

count<=3'b100;

4'b0011:

count<=3'b101;

4'b0100:

count<=3'b110;

4'b0101:

count<=3'b111;

default:

state<=INIT;

endcase

if(auth\_bit==1)

state<=wait\_for\_rng;

end

wait\_for\_rng:

begin

case(level\_num)

4'b0001:

count<=3'b011;

4'b0010:

count<=3'b100;

4'b0011:

count<=3'b101;

4'b0100:

count<=3'b110;

4'b0101:

count<=3'b111;

default:

state<=INIT;

endcase

win<=0;

if(logout==1)

state<=INIT;

else if(rng\_button==0)

state<=comp\_res;

end

comp\_res:

begin

if(logout==1)

state<=INIT;

else if(count!=0 && time\_stop==1)

begin

loose<=1;

win<=0;

state<=wait\_for\_rng;

end

else if(punch\_button==1)

begin

count<=count-3'b001;

if(count==0)

begin

if(shif\_answer!=0 && store\_reg!=0)

begin

if(shif\_answer==store\_reg)

begin

win<=1;

loose<=0;

state<=wait\_for\_level;

end

else

begin

loose<=1;

win<=0;

state<=wait\_for\_rng;

end

end

end

end

end

wait\_for\_level:

begin

win<=0;

if(levelupdated==1)

begin

state<=INIT;

end

end

endcase

end

end

endmodule

//LFSR

module LFSR(clk,rst,lfsr\_to,button\_pulse,stop);

input clk,rst,button\_pulse,stop;

//output reg lfsr\_out;

reg [10:0]count;

reg [7:0] LFSR;

output reg lfsr\_to;

reg flag;

// assign lfsr\_to = (LFSR == 11'h359);

always @(posedge clk)

begin

if (rst==0)begin

LFSR[7:0] <= 8'h00;

count =0;

lfsr\_to=0;

flag=0;

end

else

begin

if(button\_pulse==1) begin

flag=1;

end

if(stop==1)

begin

flag=0;

end

if(flag==1)begin

if(LFSR ==8'h01)begin

count=count+1'b1;

end

if(count==11'b11110110000 && (LFSR == 8'h61)) begin

lfsr\_to=1;

count=0;

end

else begin

lfsr\_to=0;

end

if (lfsr\_to) begin

LFSR[7:0] <= 8'h00;

end

else

begin

LFSR[0] <= ~LFSR[1] ^ LFSR[2] ^ LFSR[3] ^ LFSR[7];

LFSR[7:1] <= LFSR[6:0];

end

end

end

end

endmodule

//game module

//MISSION-V

//Game Module

module game\_module(twodigit,time\_stop,begin\_timer,reconfig2,input\_num,load\_time,stop,levelupdated,logout,button\_pulse,auth\_bit,punch\_button,level\_num,toggle\_answer,flash\_num,win,loose,seg\_in\_ans,clock,rst);

input clock,rst;

input button\_pulse,auth\_bit,punch\_button,logout,levelupdated;

input [3:0] level\_num;

input [3:0] toggle\_answer;

input time\_stop;

output begin\_timer,load\_time;

output [3:0] reconfig2;

output [7:0] input\_num;

output [4:0] flash\_num;

output win,loose,stop,twodigit;

output [3:0] seg\_in\_ans;

wire reg\_enable2;

wire [27:0] store\_reg;

wire time\_out,time\_in;

wire[3:0] reconfig;

wire win,loose;

RSG rsg2(win,loose,twodigit,time\_stop,begin\_timer,reconfig2,input\_num,load\_time,stop,logout,button\_pulse,auth\_bit,level\_num,time\_in,time\_out,flash\_num,reg\_enable2,store\_reg,reconfig,clock,rst);

Comparision comp1(time\_stop,levelupdated,logout,button\_pulse,level\_num,store\_reg,toggle\_answer,auth\_bit,punch\_button,reg\_enable2,win,loose,seg\_in\_ans,clock,rst);

Reconfig\_timer reconfig\_inst(clock,rst,time\_out,reconfig,time\_in);

Endmodule

//ROM Password control

// MISSION-V

// ROM password control

module ROM\_Password\_Control(rng\_button,valid\_bit, ROM\_access,auth\_button,log\_out, entered, internal\_id, address, password, auth\_bit, red\_led, green\_led, RAM\_access, password\_change,clock, rst);

//Declaring inputs and outputs

input valid\_bit,log\_out,auth\_button,ROM\_access,rng\_button;

input [2:0] internal\_id;

input [15:0] entered,password;

input clock,rst;

output reg auth\_bit,RAM\_access,password\_change;

output reg red\_led, green\_led;

output reg [2:0] address;

//register to store state

reg [2:0] state;

parameter INIT=0,ROM\_addr=1,delay1=2,delay2=3,compare=4,success=5,failed=6,halt=7;

//loop at each positive edge of the clock

always @ (posedge clock)

begin

if(rst==0)

begin

address<=3'b000;

state<=INIT;

red\_led<=0;

green\_led<=0;

auth\_bit<=0;

RAM\_access<=0;

password\_change<=0;

end

else

begin

case(state)

INIT:

begin

address<=3'b000;

red\_led<=0;

green\_led<=0;

auth\_bit<=0;

RAM\_access<=0;

password\_change<=0;

if(ROM\_access==1)

begin

state<=ROM\_addr;

end

else

begin

state<=INIT;

address<=3'b000;

end

end

ROM\_addr:

begin

if(valid\_bit==1)

begin

address<=internal\_id;

state<=delay1;

end

end

delay1:

begin

state<=delay2;

end

delay2:

begin

state<=compare;

end

compare:

begin

if(entered==password)

begin

green\_led<=1;

state<=success;

end

else

begin

state<=failed;

end

end

success:

begin

green\_led<=1;

red\_led<=0;

auth\_bit<=1;

if(log\_out==1)

begin

RAM\_access<=0;

password\_change<=0;

state<=INIT;

end

else state<=success;

if(auth\_button==1 && ROM\_access==1)

begin

RAM\_access<=1;

password\_change<=1;

state<=halt;

end

else if(rng\_button==0 && ROM\_access==1)

state<=halt;

else state<=success;

end

failed:

begin

auth\_bit<=0;

red\_led<=1;

green\_led=0;

if(auth\_button==1)

begin

state<=INIT;

end

else

state<=failed;

if(log\_out==1)

begin

RAM\_access<=0;

password\_change<=0;

state<=INIT;

end

else state<=failed;

end

halt:

begin

if(log\_out==1)

begin

RAM\_access<=0;

password\_change<=0;

state<=INIT;

end

else state<=halt;

end

endcase

end

end

endmodule

//score\_conv

//MISSION-V

//score conv

`timescale 1ps/1ps

module score\_conv(score,out\_score,clock,rst);

input clock,rst;

input [5:0] score;

output reg [7:0] out\_score;

always @(posedge clock)

begin

if(rst==0)

begin

out\_score<=8'b00000000;

end

else

begin

case(score)

6'b000000:out\_score=8'b00000000;

6'b000001:out\_score=8'b00000001;

6'b000010:out\_score=8'b00000010;

6'b000011:out\_score=8'b00000011;

6'b000100:out\_score=8'b00000100;

6'b000101:out\_score=8'b00000101;

6'b000110:out\_score=8'b00000110;

6'b000111:out\_score=8'b00000111;

6'b001000:out\_score=8'b00001000;

6'b001001:out\_score=8'b00001001;

6'b001010:out\_score=8'b00010000;

6'b001011:out\_score=8'b00010001;

6'b001100:out\_score=8'b00010010;

6'b001101:out\_score=8'b00010011;

6'b001110:out\_score=8'b00010100;

6'b001111:out\_score=8'b00010101;

6'b010000:out\_score=8'b00010110;

6'b010001:out\_score=8'b00010111;

6'b010010:out\_score=8'b00011000;

6'b010011:out\_score=8'b00011001;

6'b010100:out\_score=8'b00100000;

6'b010101:out\_score=8'b00100001;

6'b010110:out\_score=8'b00100010;

6'b010111:out\_score=8'b00100011;

6'b011000:out\_score=8'b00100100;

6'b011001:out\_score=8'b00100101;

6'b011010:out\_score=8'b00100110;

6'b011011:out\_score=8'b00100111;

6'b011100:out\_score=8'b00101000;

6'b011101:out\_score=8'b00101001;

6'b011110:out\_score=8'b00110000;

6'b011111:out\_score=8'b00110001;

6'b100000:out\_score=8'b00110010;

6'b100001:out\_score=8'b00110011;

6'b100010:out\_score=8'b00110100;

6'b100011:out\_score=8'b00110101;

6'b100100:out\_score=8'b00110110;

6'b100101:out\_score=8'b00110111;

6'b100110:out\_score=8'b00111000;

6'b100111:out\_score=8'b00111001;

6'b101000:out\_score=8'b01000000;

6'b101001:out\_score=8'b01000001;

6'b101010:out\_score=8'b01000010;

6'b101011:out\_score=8'b01000011;

6'b101100:out\_score=8'b01000100;

6'b101101:out\_score=8'b01000101;

6'b101110:out\_score=8'b01000110;

6'b101111:out\_score=8'b01000111;

6'b110000:out\_score=8'b01001000;

6'b110001:out\_score=8'b01001001;

6'b110010:out\_score=8'b01010000;

6'b110011:out\_score=8'b01010001;

6'b110100:out\_score=8'b01010010;

6'b110101:out\_score=8'b01010011;

6'b110110:out\_score=8'b01010100;

6'b110111:out\_score=8'b01010101;

6'b111000:out\_score=8'b01010110;

6'b111001:out\_score=8'b01010111;

6'b111010:out\_score=8'b01011000;

6'b111011:out\_score=8'b01011001;

6'b111100:out\_score=8'b01100000;

6'b111101:out\_score=8'b01100001;

6'b111110:out\_score=8'b01100010;

6'b111111:out\_score=8'b01100011;

endcase

end

end

endmodule

//score top

//MISSION-V

//score top module

`timescale 1ps/1ps

module score\_table(clock,rst,log\_out,green\_user,internal\_id,auth\_bit,win,loose,disp\_button,level\_num,disp\_out,green\_max);

input clock,rst,green\_user,auth\_bit,win,loose,disp\_button,log\_out;

input [3:0] level\_num;

input [2:0] internal\_id;

output [7:0] disp\_out;

output green\_max;

wire [2:0] address;

wire [5:0] data,q,disp;

wire wren;

score\_controller scorecont1(clock,rst,green\_user,internal\_id,auth\_bit,log\_out,level\_num,win,loose,address,q,data,wren,disp\_button,disp,green\_max);

RAM\_score\_module ramscore1(address,clock,data,wren,q);

score\_conv scon1(disp,disp\_out,clock,rst);

endmodule

//4 bit binary counter

// ECE6370

// Author: Manasa Tempalli 5355

//4 bit binary counter

module four\_Bit\_Binary\_Counter(button\_inv,clk,rst,random\_num,enable);

input button\_inv,clk,rst;

output reg [3:0] random\_num;

output reg enable;

reg [3:0] count;

reg flag;

always @(posedge clk)

begin

if(rst==0)

begin

count<=4'b0000;

random\_num<=4'b0000;

enable<=0;

flag=0;

end

else

begin

if(button\_inv==1)

begin

flag<=1;

enable<=0;

count<=count+4'b0001;

end

else

begin

if(flag==1)

begin

flag<=0;

enable<=1;

random\_num<=count;

end

else enable<=0;

end

end

end

endmodule

//ROM\_userID\_control

//MISSION-V

//Module for ROM User ID Control

`timescale 1ps/1ps

module ROM\_User\_ID\_Control(entered,log\_out,valid\_bit,status,address,user\_id,internal\_id,ROM\_access,RAM\_access,green\_led\_user,red\_led\_user,clock,rst);

input[15:0] entered,user\_id;

input [6:0] status;

output reg [2:0] internal\_id,address;

output reg ROM\_access,RAM\_access,red\_led\_user,green\_led\_user;

input valid\_bit,log\_out;

input clock,rst;

reg[2:0] state;

parameter INIT=0,ROM\_addr=1,delay1=2,delay2=3,comparing=4,ROM\_RAM\_access=5,fail=6,halt=7;

always @(posedge clock)

begin

if(rst==0)

begin

red\_led\_user<=0;

green\_led\_user<=0;

state<=INIT;

address<=3'b111;

ROM\_access<=0;

RAM\_access<=0;

internal\_id<=3'b000;

end

else

begin

case(state)

INIT:

begin

red\_led\_user<=0;

green\_led\_user<=0;

address<=3'b111;

ROM\_access<=0;

RAM\_access<=0;

internal\_id<=3'b000;

if(valid\_bit==1)

begin

state<=ROM\_addr;

end

else state<=INIT;

end

ROM\_addr:

begin

if(address!=3'b110)

begin

address<=address+3'b001;

state<=delay1;

end

else begin

state<=fail;

red\_led\_user<=1;

end

end

delay1:

begin

state<=delay2;

end

delay2:

begin

state<=comparing;

end

comparing:

begin

if(entered==user\_id)

begin

green\_led\_user<=1;

internal\_id<=address;

state<=ROM\_RAM\_access;

end

else

begin

red\_led\_user<=1;

state<=ROM\_addr;

end

end

ROM\_RAM\_access:

begin

green\_led\_user<=1;

red\_led\_user<=0;

if(log\_out==1)

begin

ROM\_access<=0;

RAM\_access<=0;

state<=INIT;

green\_led\_user<=0;

end

else begin

state<=ROM\_RAM\_access;

end

if(status[internal\_id]==1)

begin

RAM\_access<=1;

state<=halt;

end

else

begin

ROM\_access<=1;

state<=halt;

end

end

fail:

begin

red\_led\_user<=1;

if(log\_out==1)

begin

ROM\_access<=0;

RAM\_access<=0;

green\_led\_user<=0;

state<=INIT;

end

else

state<=fail;

end

halt:

begin

if(log\_out==1)

begin

ROM\_access<=0;

RAM\_access<=0;

state<=INIT;

green\_led\_user<=0;

end

else

state<=halt;

end

endcase

end

end

endmodule

//two digit timer

module two\_digit\_timer(clock,rst,twodigit,begin\_timer,load\_time,reconfig2,input\_num,time\_stop,timedig1,timedig2,stop,b);

input clock,rst;

input begin\_timer,load\_time,stop,twodigit;

input [3:0] reconfig2;

input [7:0] input\_num;

output time\_stop;

output [3:0] timedig1,timedig2;

wire outpulse,borrow,stop\_upstream;

output b;

Reconfig\_timer reconinst2(clock,rst,begin\_timer,reconfig2,outpulse,stop);

timer\_digit2 timedig11(outpulse,load\_time,input\_num[3:0],timedig1,stop\_upstream,time\_stop,borrow,clock,rst);

timer\_digit2 timedig22(borrow,load\_time,input\_num[7:4],timedig2,twodigit,stop\_upstream,b,clock,rst);

endmodule

//Button inverter

// ECE6370

// Author: Manasa Tempalli 5355

// Button Inverter

module Button\_Inverter(button\_pulse,auth\_bit,clk,rst,button\_inv);

input button\_pulse,clk,rst,auth\_bit;

output reg button\_inv;

always @ (posedge clk)

begin

if(rst==0)

button\_inv<=0;

else

begin

if(auth\_bit==1)

button\_inv<=~(button\_pulse);

else button\_inv<=0;

end

end

endmodule

//load register

// ECE6370

// Author: Manasa Tempalli 5355

// Register for storing and loading

module load\_register(logout,I,Q,ld,enable,clk,rst);

input[3:0] I;

input ld,clk,rst,enable,logout;

output[3:0] Q;

reg[3:0] R;

always @ (posedge clk)

begin

//Register bits set to '0' when reset is set

if(rst==0)

begin

R<=4'b0000;

end

//storing to the register when load button set to '1'

else if(enable==1)

begin

if(logout==1)

R<=4'b0000;

if(ld==1)

begin

R<=I;

end

end

else R<=4'b0000;

end

//assigning R to Q

assign Q=R;

endmodule

//ROMPassword top module

//MISSION-V

//ROM password top module

module ROM\_Password\_Top\_Module(ROM\_access,rng\_button,auth\_button,log\_out,toggle\_entry,internal\_id,auth\_bit,red\_led,green\_led,RAM\_access,password\_change,clock,rst);

input auth\_button,log\_out,ROM\_access,rng\_button;

input[3:0] toggle\_entry;

input [2:0] internal\_id;

input clock,rst;

output auth\_bit, red\_led, green\_led, RAM\_access, password\_change;

wire [2:0] address;

wire [15:0] password;

wire valid\_bit;

wire [15:0] entered;

Shift\_Reg shif\_rompass1(ROM\_access,ROM\_access,toggle\_entry,auth\_button,log\_out,entered,valid\_bit,clock,rst);

ROM\_Password\_Control ROM\_Password\_control1(rng\_button,valid\_bit, ROM\_access,auth\_button,log\_out, entered, internal\_id, address, password, auth\_bit, red\_led, green\_led, RAM\_access, password\_change,clock, rst);

ROM\_Password\_Module ROM\_Password1(address,clock,password);

endmodule

//Comparision module

//MISSION-V

//Module for getting results and compare

module Comparision(time\_stop,levelupdated,logout,rng\_button,level\_num,store\_reg,toggle\_answer,auth\_bit,punch\_button,reg\_enable2,win,loose,seg\_in\_ans,clock,rst);

input logout,rng\_button,auth\_bit,punch\_button,reg\_enable2,clock,rst,levelupdated,time\_stop;

input [3:0] toggle\_answer;

input [27:0] store\_reg;

input[3:0] level\_num;

output win,loose;

output [3:0] seg\_in\_ans;

wire [27:0] shif\_answer;

result\_comp res1(time\_stop,levelupdated,logout,rng\_button,auth\_bit,punch\_button,shif\_answer,store\_reg,level\_num,win,loose,clock,rst);

shift\_reg\_28 shl28reg(toggle\_answer,reg\_enable2,punch\_button,shif\_answer,clock,rst);

load\_register loadinst(logout,toggle\_answer,seg\_in\_ans,punch\_button,reg\_enable2,clock,rst);

endmodule

//button shaper

// ECE6370

// Author: Manasa Tempalli 5355

// Single pulse generator for Button push

module button\_shaper(button\_push, button\_pulse, clk, rst);

//declaring inputs and outputs

input button\_push;

output button\_pulse;

reg button\_pulse;

//clock and reset

input clk,rst;

//register to store state

reg[2:0] state,state\_next;

parameter s\_wait=0, state1=1, state2=2, state3=3;

//loop at each positive edge of clock cycle

always @ (state, button\_push)

begin

case(state)

s\_wait:

begin

button\_pulse<=0;

if (button\_push==1)

state\_next<=s\_wait;

else

//moving to pulse generation states when button pushed

state\_next<=state1;

end

//Single pulse generation

state1:

begin

button\_pulse<=1;

if(button\_push==1)

state\_next<=s\_wait;

else

state\_next<=state2;

end

state2:

begin

button\_pulse<=0;

if(button\_push==1)

state\_next<=s\_wait;

else

state\_next<=state2;

end

default:

state\_next<=s\_wait;

endcase

end

always @ (posedge clk)

begin

if(rst==0)

state <= s\_wait;

else

state <= state\_next;

end

endmodule

//seven segment

// ECE6370

// Author: Manasa Tempalli 5355

//Seven segment display mapping

module seven\_seg(seg\_in, seg\_out);

//declaring inputs and outputs

input[3:0] seg\_in;

output[6:0] seg\_out;

reg[6:0] seg\_out;

//loop for sensitivity list

always @ (seg\_in)

begin //mapping inputs to the display leds begins

//'0' turns LED On and '1' turns LED Off

case(seg\_in)

4'h0:

begin

seg\_out=7'b0000001;

//displays '0'

end

4'h1:

begin

seg\_out=7'b1001111;

//displays '1'

end

4'h2:

begin

seg\_out=7'b0010010;

//displays '2'

end

4'h3:

begin

seg\_out=7'b0000110;

//displays '3'

end

4'h4:

begin

seg\_out=7'B1001100;

//displays '4'

end

4'h5:

begin

seg\_out=7'b0100100;

//displays '5'

end

4'h6:

begin

seg\_out=7'b0100000;

//displays '6'

end

4'h7:

begin

seg\_out=7'b0001111;

//displays '7'

end

4'h8:

begin

seg\_out=7'b0000000;

//displays '8'

end

4'h9:

begin

seg\_out=7'b0000100;

//displays '9'

end

4'hA:

begin

seg\_out=7'b0000010;

//displays 'a'

end

4'hB:

begin

seg\_out=7'b1100000;

//displays 'b'

end

4'hC:

begin

seg\_out=7'b0110001;

//displays 'C'

end

4'hD:

begin

seg\_out=7'b1000010;

//displays 'd'

end

4'hE:

begin

seg\_out=7'b0010000;

//displays 'e'

end

4'hF:

begin

seg\_out=7'b0111000;

//displays 'F'

end

default:

begin

seg\_out=7'b1111111;

end

endcase

end

endmodule

//level top module

//MISSION-V

//level tabel top module

`timescale 1ps/1ps

module level\_top\_module(levelupdated,green\_user,rng\_button,log\_out,auth\_bit,win,internal\_id,level\_num,clock,rst);

//Declaring inputs and outputs

input clock,rst;

input rng\_button,log\_out,auth\_bit,win,green\_user;

input [2:0] internal\_id;

output [3:0] level\_num;

output levelupdated;

wire [2:0] address,data,q;

wire wren;

Level\_Controller level\_con1(levelupdated,green\_user,rng\_button,log\_out,internal\_id,auth\_bit,win,address,q,data,wren,level\_num,clock,rst);

level\_table level\_tab1(address,clock,data,wren,q);

endmodule

//ROM User ID top

//MISSION-V

//ROM user id top module

module ROM\_User\_ID\_Top\_Module(toggle\_entry,auth\_button,status,internal\_id,ROM\_access,RAM\_access,green\_led\_user,red\_led\_user,log\_out,clock,rst);

input [3:0] toggle\_entry;

input [6:0] status;

input auth\_button,log\_out;

input clock,rst;

output [2:0] internal\_id;

output ROM\_access,RAM\_access,red\_led\_user,green\_led\_user;

wire [15:0] user\_id,entered;

wire [2:0] address;

wire valid\_bit;

Shift\_Reg shift1(auth\_button,auth\_button,toggle\_entry,auth\_button,log\_out,entered,valid\_bit,clock,rst);

ROM\_User\_ID\_Control ROM\_user\_control1(entered,log\_out,valid\_bit,status,address,user\_id,internal\_id,ROM\_access,RAM\_access,green\_led\_user,red\_led\_user,clock,rst);

ROM\_User\_ID\_Module ROM\_User\_1(address,clock,user\_id);

Endmodule

//level controller

//MISSION-V

//Level\_Controller

`timescale 1ps/1ps

module Level\_Controller(levelupdated,green\_user,rng\_button,log\_out,internal\_id,auth\_bit,win,address,level\_i,level\_o,wren,level\_num,clk,rst);

//Decalring input and output signals

input [2:0] internal\_id,level\_i;

input auth\_bit,clk,rst,rng\_button,log\_out,green\_user;

input win;

output reg [2:0] address,level\_o;

output reg [3:0] level\_num;

output reg wren,levelupdated;

//Declaring state register and the state values

reg [3:0] state;

reg [2:0] level;

parameter INIT=0,wait1=1,wait2=2,level\_check=3,level\_inc=4,level\_update=5,win\_wait=6,halt=7,halt2=8;

//Begin functionality procedure

always @(posedge clk)

begin

if(rst==0)

begin

address<=3'b000;

level\_o<=3'b000;

level\_num<=4'b0000;

level<=3'b000;

state<=INIT;

end

else

begin

case(state)

INIT:

begin

address<=3'b000;

wren<=0;

level\_o<=3'b000;

level\_num<=4'b0000;

level<=3'b000;

if(green\_user==1)

begin

address<=internal\_id;

state<=wait1;

end

end

wait1:

begin

wren<=0;

if(log\_out==1)

begin

state<=halt2;

end

else state<=wait2;

end

wait2:

begin

if(log\_out==1)

begin

state<=halt2;

end

else state<=level\_check;

end

level\_check:

begin

if(log\_out==1)

begin

state<=halt2;

end

else if(level\_i==3'b000)

begin

level<=level\_i;

state<=level\_inc;

end

else if(level==3'b110)

begin

state<=halt;

end

else

begin

level<=level\_i;

level\_num<={1'b0,level\_i};

levelupdated<=1;

if(auth\_bit==1)

state<=win\_wait;

end

end

level\_inc:

begin

if(log\_out==1)

begin

state<=halt2;

end

else

begin

level<=level+3'b001;

wren<=1;

state<=level\_update;

end

end

level\_update:

begin

if(log\_out==1)

begin

state<=halt2;

end

else if(level!=3'b110)

begin

level\_o<=level;

state<=wait1;

end

else state<=halt;

end

win\_wait:

begin

levelupdated<=0;

if(log\_out==1)

begin

state<=halt2;

end

else if(level==3'b101 && win==1)

state<=halt;

else if(win==1 && level!=3'b101)

begin

state<=level\_inc;

end

end

halt:

begin

if(log\_out==1)

begin

state<=halt2;

end

else if(rng\_button==1)

begin

wren<=1;

level\_o<=3'b001;

state<=INIT;

end

end

halt2:

begin

if(green\_user==0)

state<=INIT;

end

default:

begin

state<=INIT;

end

endcase

end

end

endmodule

//MISSION-V

//RAM Password Reset Top Module

module RAM\_Password\_Reset\_Top\_Module(RAM\_access1,RAM\_access2,password\_change,auth\_button,toggle\_entry,internal\_id,status,log\_out,green\_led,red\_led,auth\_bit,clock,rst);

input RAM\_access1,RAM\_access2,password\_change,log\_out,auth\_button;

input [3:0] toggle\_entry;

input [2:0] internal\_id;

input clock,rst;

output [6:0] status;

output green\_led,red\_led,auth\_bit;

wire [2:0] address;

wire [15:0] password,data;

wire wren,valid\_bit;

wire [15:0] entered;

Shift\_Reg shif\_ram\_1(RAM\_access1,RAM\_access2,toggle\_entry,auth\_button,log\_out,entered,valid\_bit,clock,rst);

RAM\_access\_control RAM\_control1(RAM\_access1,RAM\_access2,password\_change,internal\_id,auth\_button,wren,entered,valid\_bit,data,address,password,status,log\_out,green\_led,red\_led,auth\_bit,clock,rst);RAM\_Password\_Reset\_Module RAM\_module1(address,clock,data,wren,password);

Endmodule

For 5 bit input seven segment display

// ECE6370

// Author: Manasa Tempalli 5355

//Seven segment display mapping

module seven\_seg5(seg\_in, seg\_out);

//declaring inputs and outputs

input[4:0] seg\_in;

output[6:0] seg\_out;

reg[6:0] seg\_out;

//loop for sensitivity list

always @ (seg\_in)

begin //mapping inputs to the display leds begins

//'0' turns LED On and '1' turns LED Off

case(seg\_in)

5'b00000:

begin

seg\_out=7'b0000001;

//displays '0'

end

5'b00001:

begin

seg\_out=7'b1001111;

//displays '1'

end

5'b00010:

begin

seg\_out=7'b0010010;

//displays '2'

end

5'b00011:

begin

seg\_out=7'b0000110;

//displays '3'

end

5'b00100:

begin

seg\_out=7'B1001100;

//displays '4'

end

5'b00101:

begin

seg\_out=7'b0100100;

//displays '5'

end

5'b00110:

begin

seg\_out=7'b0100000;

//displays '6'

end

5'b00111:

begin

seg\_out=7'b0001111;

//displays '7'

end

5'b01000:

begin

seg\_out=7'b0000000;

//displays '8'

end

5'b01001:

begin

seg\_out=7'b0000100;

//displays '9'

end

5'b01010:

begin

seg\_out=7'b0000010;

//displays 'a'

end

5'b01011:

begin

seg\_out=7'b1100000;

//displays 'b'

end

5'b01100:

begin

seg\_out=7'b0110001;

//displays 'C'

end

5'b01101:

begin

seg\_out=7'b1000010;

//displays 'd'

end

5'b01110:

begin

seg\_out=7'b0010000;

//displays 'e'

end

5'b01111:

begin

seg\_out=7'b0111000;

//displays 'F'

end

default:

begin

seg\_out=7'b1111111;

end

endcase

end

endmodule

//MISSION-V

//shift register for 28 bit sequence storage

module shift\_reg\_28(flash\_num,reg\_enable,shl,Q,clock,rst);

input shl,reg\_enable,rst,clock;

input [3:0] flash\_num;

output [27:0] Q;

reg [27:0] R;

always @(posedge clock)

begin

if(rst==0)

R<=28'b0;

else if (reg\_enable==1)

begin

if(shl==1)

begin

R[3:0]<=flash\_num;

R[7:4]<=R[3:0];

R[11:8]<=R[7:4];

R[15:12]<=R[11:8];

R[19:16]<=R[15:12];

R[23:20]<=R[19:16];

R[27:24]<=R[23:20];

end

else R<=R;

end

else R<=28'b0;

end

assign Q=R;

endmodule

//MISSION-V

//Random Sequence Generator

module RSG(win,loose,twodigit,time\_stop,begin\_timer,reconfig2,input\_num,load\_time,stop,logout,button\_pulse,auth\_bit,level\_num,time\_in,time\_out,flash\_num,reg\_enable2,store\_reg,reconfig,clock,rst);

input logout,button\_pulse,clock,rst,time\_in,auth\_bit;

input [3:0] level\_num;

input time\_stop;

input win,loose;

output begin\_timer,load\_time;

output [3:0] reconfig2;

output [7:0] input\_num;

output wire [4:0] flash\_num;

output time\_out,stop,twodigit;

output reg\_enable2;

output [27:0] store\_reg;

output [3:0] reconfig;

wire [3:0] random\_num;

wire enable,reg\_enable,shl;

wire [4:0] address,q;

RNG rn1(button\_pulse,auth\_bit,clock,rst,random\_num,enable);

Random\_seq rs1(win,loose,twodigit,time\_stop,begin\_timer,reconfig2,input\_num,load\_time,logout,random\_num,enable,level\_num,time\_in,time\_out,stop,address,q,reg\_enable,reg\_enable2,shl,flash\_num,reconfig,clock,rst);

ROM\_seq romseq1(address,clock,q);

shift\_reg\_28 shl28(flash\_num,reg\_enable,shl,store\_reg,clock,rst);

endmodule

// ECE6370

// Author: Manasa Tempalli 5355

//Random Number Generator

`timescale 1ps/1ps

module RNG(button\_pulse,auth\_bit,clk,rst,random\_num,enable);

input button\_pulse,clk,rst,auth\_bit;

output [3:0] random\_num;

output enable;

wire button\_inv;

Button\_Inverter button\_inv1(button\_pulse,auth\_bit,clk,rst,button\_inv);

four\_Bit\_Binary\_Counter binary\_counter1(button\_inv,clk,rst,random\_num,enable);

endmodule

//MISSION-V

//Random Sequence module

`timescale 1 ps/1 ps

module Random\_seq(win,loose,twodigit,time\_stop,begin\_timer,reconfig2,input\_num,load\_time,logout,random\_num,enable,level\_num,time\_in,time\_out,stop,address,q,reg\_enable,reg\_enable2,shl,flash\_num,reconfig,clock,rst);

//Declaring inputs and outputs

input [3:0] random\_num;

input [3:0] level\_num;

input enable,time\_in,logout,time\_stop;

input clock,rst;

input [4:0] q;

output reg[4:0] address;

output reg time\_out,reg\_enable,reg\_enable2,shl,stop,twodigit;

output reg [4:0] flash\_num;

output reg [3:0] reconfig,reconfig2;

output reg begin\_timer,load\_time;

output reg [7:0] input\_num;

input win,loose;

reg [2:0] count;

reg [3:0] state;

reg [4:0] gap;

parameter INIT=0,seq\_ON=1,wait1=2,wait2=3,get\_seq=4,wait\_for\_sec=5,flash0=6,wait\_for\_sec2=7,timer=8,halt=9;

always @(posedge clock)

begin

if(rst==0)

begin

state<=INIT;

address<=5'b00000;

time\_out<=0;

flash\_num<=5'b11111;

gap<=5'b00000;

count<=3'b000;

reg\_enable<=0;

reg\_enable2<=0;

input\_num<=8'b00000000;

begin\_timer<=0;

load\_time<=0;

reconfig2<=4'b1010;

twodigit<=1;

end

else

begin

case(state)

INIT:

begin

address<=5'b00000;

time\_out<=0;

flash\_num<=5'b11111;

gap<=5'b00000;

count<=3'b000;

reg\_enable<=0;

reg\_enable2<=0;

begin\_timer<=0;

load\_time<=1;

reconfig2<=4'b1010;

stop<=0;

twodigit<=1;

if(enable==1)

begin

reg\_enable<=0;

reg\_enable2<=0;

address<={1'b0,random\_num+flash\_num[3:0]};

gap<={1'b0,random\_num};

case(level\_num)

3'b001:

begin

count<=3'b011;

input\_num<=8'b00110000;

end

3'b010:

begin

count<=3'b100;

input\_num<=8'b00110101;

end

3'b011:

begin

count<=3'b101;

input\_num<=8'b01000000;

end

3'b100:

begin

count<=3'b110;

input\_num<=8'b01000101;

end

3'b101:

begin

count<=3'b111;

input\_num<=8'b01010000;

end

default:

count<=3'b000;

endcase

state<=seq\_ON;

end

end

seq\_ON:

begin

load\_time<=0;

reg\_enable<=1;

reg\_enable2<=1;

count<=count-3'b001;

if(logout==1)

begin

state<=INIT;

reg\_enable<=0;

reg\_enable2<=0;

end

if(count==0)

begin

begin\_timer<=1;

state<=timer;

end

else

begin

address<=address+gap+level\_num+flash\_num[3:0];

state<=wait1;

end

end

wait1:

begin

state<=wait2;

if(logout==1)

begin

state<=INIT;

reg\_enable<=0;

reg\_enable2<=0;

end

end

wait2:

begin

state<=get\_seq;

if(logout==1)

begin

state<=INIT;

reg\_enable<=0;

reg\_enable2<=0;

end

end

get\_seq:

begin

flash\_num<={1'b0,q[3:0]};

shl<=1;

time\_out<=1;

reconfig<=4'b1010;

state<=wait\_for\_sec;

if(logout==1)

begin

state<=INIT;

reg\_enable<=0;

reg\_enable2<=0;

end

end

wait\_for\_sec:

begin

time\_out<=0;

shl<=0;

if(logout==1)

begin

state<=INIT;

reg\_enable<=0;

reg\_enable2<=0;

end

if(time\_in==1)

begin

state<=flash0;

end

end

flash0:

begin

flash\_num<=5'b11111;

time\_out<=1;

reconfig<=4'b1010;

state<=wait\_for\_sec2;

if(logout==1)

begin

state<=INIT;

reg\_enable<=0;

reg\_enable2<=0;

end

end

wait\_for\_sec2:

begin

time\_out<=0;

if(logout==1)

begin

state<=INIT;

reg\_enable<=0;

reg\_enable2<=0;

end

if(time\_in==1)

begin

state<=seq\_ON;

end

end

timer:

begin

if(win==1||loose==1)

begin

stop<=1;

input\_num<=8'b00000000;

reconfig2<=4'b1010;

reg\_enable<=0;

reg\_enable2<=0;

state<=halt;

end

else if(logout==1)

begin

state<=INIT;

input\_num<=8'b00000000;

reg\_enable<=0;

reg\_enable2<=0;

end

load\_time<=0;

flash\_num<=5'b11111;

begin\_timer<=0;

if(time\_stop==1)

begin

stop<=1;

input\_num<=8'b00000000;

reconfig2<=4'b1010;

reg\_enable<=0;

reg\_enable2<=0;

state<=halt;

end

end

halt:

state<=INIT;

endcase

end

end

endmodule

//timer didgits module

module timer\_digit2(decrement,load,input\_num,output\_num,stop\_upstream,stop\_downstream,borrow,clock,rst);

input decrement,load;

input clock,rst;

input [3:0] input\_num;

input stop\_upstream;

output [3:0] output\_num;

output stop\_downstream,borrow;

reg[3:0] num;

reg flag,flag2;

reg stop\_down,b;

always @(posedge clock)

begin

if(rst==0)

begin

stop\_down=0;

b<=0;

end

else

begin

if(load==1)

begin

stop\_down<=0;

num<=input\_num;

end

if(decrement==1)

begin

if(num==4'b0001)

begin

if(stop\_upstream==1)

begin

flag2<=1;

end

end

if(num==4'b0000)

begin

if(stop\_upstream==1)

begin

stop\_down<=1;

b<=0;

end

else

begin

b<=1;

end

end

flag<=1;

end

else flag<=0;

if(flag==1)

begin

if(flag2==1)

begin

stop\_down<=1;

flag2<=0;

end

if(num==4'b0000)

begin

b<=0;

if(stop\_upstream==1)

begin

num<=4'b0000;

end

else

begin

num<=4'b1001;

end

end

else if(num!=4'b0000)

begin

b<=0;

num<=num-4'b0001;

end

end

end

end

assign output\_num=num;

assign stop\_downstream=stop\_down;

assign borrow=b;

endmodule

//Shift register

//MISSION-V

//Shift\_Register

module Shift\_Reg(enable1,enable2,toggle\_entry,auth\_button,log\_out,entered,valid\_bit,clock,rst);

input[3:0] toggle\_entry;

input clock,rst,auth\_button,log\_out,enable1,enable2;

output reg [15:0] entered;

output reg valid\_bit;

reg [1:0] count;

reg [15:0] R;

reg[2:0] state;

parameter INIT=0,button\_wait=1,load=2,enter=3,halt=4;

always @(posedge clock)

begin

if(rst==0)

begin

state<=INIT;

R<=16'b0000000000000000;

entered<=16'b0000000000000000;

count<=2'b11;

valid\_bit<=0;

end

else

begin

case(state)

INIT:

begin

R<=16'b0000000000000000;

entered<=16'b0000000000000000;

count<=2'b00;

valid\_bit<=0;

if(enable1==1 || enable2==1)

begin

state<=button\_wait;

end

else

state<=INIT;

end

button\_wait:

begin

if(auth\_button==1)

begin

count<=count+2'b01;

state<=load;

end

else state<=button\_wait;

end

load:

begin

R[15:12]<=R[11:8];

R[11:8]<=R[7:4];

R[7:4]<=R[3:0];

R[3:0]<=toggle\_entry;

if(count==2'b00)

begin

state<=enter;

end

else

begin

state<=button\_wait;

end

end

enter:

begin

valid\_bit<=1;

entered<=R;

state<=halt;

if(log\_out==1)

begin

valid\_bit<=0;

state<=INIT;

end

else state<=enter;

end

halt:

begin

if(log\_out==1)

begin

valid\_bit<=0;

state<=INIT;

end

else state<=halt;

end

endcase

end

end

endmodule

//score controller

//MISSION-V

//controller for RAM storing scores. Score controller

module score\_controller(clock,rst,green\_user,internal\_id,auth\_bit,log\_out,level\_num,win,loose,address,q,data,wren,disp\_button,disp,green\_max);

input clock,rst;

input [2:0] internal\_id;

input [3:0] level\_num;

input green\_user,auth\_bit,win,loose,log\_out,disp\_button;

input [5:0] q;

output reg wren,green\_max;

output reg [2:0] address;

output reg [5:0] data;

output reg [5:0] disp;

reg [5:0] ind\_score,team\_max;

reg [3:0] state;

parameter INIT=0,wait1=1,wait2=2,read\_team\_max=3,wait3=4,wait4=5,read\_ind\_score=6,wait\_for\_win=7,update\_RAM=8,wait5=9,wait6=10,wait7=11,update\_team\_max=12;

always @(posedge clock)

begin

if(rst==0)

begin

address<=3'b000;

disp<=6'b000000;

ind\_score<=6'b000000;

team\_max<=6'b000000;

wren<=0;

green\_max<=0;

state<=INIT;

end

else

begin

case(state)

INIT:

begin

address<=3'b111;

disp<=6'b000000;

ind\_score<=6'b000000;

team\_max<=6'b000000;

wren<=0;

green\_max<=0;

state<=wait1;

end

wait1:

begin

state<=wait2;

end

wait2:

begin

state<=read\_team\_max;

end

read\_team\_max:

begin

team\_max<=q;

if(green\_user==1)

begin

address<=internal\_id;

state<=wait3;

end

end

wait3:

begin

state<=wait4;

end

wait4:

begin

state<=read\_ind\_score;

end

read\_ind\_score:

begin

ind\_score<=q;

if(auth\_bit==1)

state<=wait\_for\_win;

end

wait\_for\_win:

begin

wren<=1;

if(log\_out==1)

begin

state<=update\_RAM;

end

else if(win==1)

begin

ind\_score<=ind\_score+level\_num;

end

else if(loose==1)

begin

ind\_score<=6'b000000;

end

if(ind\_score>=team\_max)

begin

green\_max<=1;

team\_max<=ind\_score;

end

else green\_max<=0;

if(disp\_button==1)

begin

disp<=ind\_score;

end

else disp<=team\_max;

end

update\_RAM:

begin

data<=ind\_score;

state<=wait5;

end

wait5:

begin

address<=3'b111;

state<=wait6;

end

wait6:

begin

state<=wait7;

end

wait7:

begin

state<=update\_team\_max;

end

update\_team\_max:

begin

data<=team\_max;

state<=INIT;

end

endcase

end

end

endmodule

//RAM access control

//MISSION-V

//RAM access control

module RAM\_access\_control(RAM\_access1,RAM\_access2,password\_change,internal\_id,auth\_button,wren,entered,valid\_bit,data,address,password,status,log\_out,green\_led,red\_led,auth\_bit,clock,rst);

input RAM\_access1,RAM\_access2,password\_change,valid\_bit,log\_out,auth\_button;

input clock,rst;

input[2:0] internal\_id;

input [15:0] entered,password;

output reg[15:0] data;

output reg[2:0] address;

output reg [6:0] status;

output reg red\_led,green\_led,auth\_bit,wren;

reg[3:0] state;

parameter INIT=0,password\_update=1,update=2,update\_done=3,password\_check=4,RAM\_addr=5,delay1=6,delay2=7,compare=8,success=9,fail=10,halt=11;

always @(posedge clock)

begin

if(rst==0)

begin

state<=INIT;

red\_led<=0;

green\_led<=0;

auth\_bit<=0;

address<=3'b000;

data<=16'b0000000000000000;

status<=7'b0000000;

end

else

begin

case(state)

INIT:

begin

red\_led<=0;

green\_led<=0;

auth\_bit<=0;

address<=3'b000;

wren<=0;

if(RAM\_access2==1 && password\_change==1)

begin

state<=password\_update;

end

else if(RAM\_access1==1 && password\_change==0)

begin

state<=password\_check;

end

else

state<=INIT;

end

password\_update:

begin

address<=internal\_id;

wren<=password\_change;

state<=update;

end

update:

begin

if(valid\_bit==1)

begin

data<=entered;

state<=update\_done;

end

end

update\_done:

begin

green\_led<=1;

status[internal\_id]<=1;

state<=halt;

if(log\_out==1)

begin

state<=INIT;

end

end

password\_check:

begin

if(valid\_bit==1)

begin

state<=RAM\_addr;

end

end

RAM\_addr:

begin

address<=internal\_id;

state<=delay1;

end

delay1:

begin

state<=delay2;

end

delay2:

begin

state<=compare;

end

compare:

begin

if(entered==password)

begin

state<=success;

end

else

begin

state<=fail;

end

end

success:

begin

green\_led<=1;

red\_led<=0;

auth\_bit<=1;

state<=halt;

if(log\_out==1)

begin

wren<=0;

state<=INIT;

end

end

fail:

begin

red\_led<=1;

green\_led<=0;

if(log\_out==1)

begin

wren<=0;

state<=INIT;

end

if(auth\_button==1)

begin

state<=password\_check;

end

end

halt:

begin

if(log\_out==1)

begin

state<=INIT;

end

end

endcase

end

end

endmodule

//Project Top Module

//MISSION-V

//Top module of the project

module Project\_Top\_Module(auth\_button,log\_out,rng\_button,disp\_button,toggle\_user,toggle\_pass,toggle\_answer,red\_led\_user,green\_led\_user,red\_led\_rom,green\_led\_rom,red\_led\_ram,green\_led\_ram,green\_max,loose,flash\_seg,level\_seg,seg\_answer,seg\_score\_high,seg\_score\_low,time\_show\_high,time\_show\_low,b,clock,rst);

input auth\_button,log\_out,rng\_button,disp\_button;

input [3:0] toggle\_user,toggle\_pass,toggle\_answer;

input clock,rst;

output red\_led\_user,green\_led\_user,red\_led\_rom,green\_led\_rom,red\_led\_ram,green\_led\_ram,loose,green\_max,b;

output [6:0] flash\_seg,level\_seg,seg\_answer,seg\_score\_high,seg\_score\_low,time\_show\_high,time\_show\_low;

wire auth\_button\_pulse,logout\_pulse;

wire [15:0] entered\_user,entered\_pass;

wire valid\_bit\_user,valid\_bit\_pass,auth\_bit,win,loose;

wire [2:0] internal\_id;

wire ROM\_access,RAM\_access1,RAM\_access2,password\_change,auth\_bit1,auth\_bit2;

wire[6:0] status;

wire [3:0] level\_num;

wire[3:0] seg\_in\_ans;

wire [4:0] flash\_num;

wire rng\_button\_pulse,green\_led\_user,levelupdated;

wire [7:0] disp\_out;

wire begin\_timer,load\_time;

wire [3:0] reconfig2;

wire [7:0] input\_num;

wire time\_stop,stop,twodigit;

wire [3:0] timedig1,timedig2;

//Button shaping

button\_shaper b1(auth\_button, auth\_button\_pulse, clock, rst);

button\_shaper b2(log\_out, logout\_pulse, clock, rst);

button\_shaper b3(rng\_button, rng\_button\_pulse, clock, rst);

//ROM User authentication

ROM\_User\_ID\_Top\_Module rom\_user2(toggle\_user,auth\_button\_pulse,status,internal\_id,ROM\_access,RAM\_access1,green\_led\_user,red\_led\_user,logout\_pulse,clock,rst);

ROM\_Password\_Top\_Module rom\_pass2(ROM\_access,rng\_button,auth\_button\_pulse,logout\_pulse,toggle\_pass,internal\_id,auth\_bit1,red\_led\_rom,green\_led\_rom,RAM\_access2,password\_change,clock,rst);

RAM\_Password\_Reset\_Top\_Module ram\_pass2(RAM\_access1,RAM\_access2,password\_change,auth\_button\_pulse,toggle\_pass,internal\_id,status,logout\_pulse,green\_led\_ram,red\_led\_ram,auth\_bit2,clock,rst);

or(auth\_bit,auth\_bit1,auth\_bit2);

//Level table integration with game module

game\_module gameleveltest(twodigit,time\_stop,begin\_timer,reconfig2,input\_num,load\_time,stop,levelupdated,logout\_pulse,rng\_button,auth\_bit,auth\_button\_pulse,level\_num,toggle\_answer,flash\_num,win,loose,seg\_in\_ans,clock,rst);

level\_top\_module levelgametest(levelupdated,green\_led\_user,auth\_button\_pulse,logout\_pulse,auth\_bit,win,internal\_id,level\_num,clock,rst);

//setting timer

two\_digit\_timer twodigtimeinst(clock,rst,twodigit,begin\_timer,load\_time,reconfig2,input\_num,time\_stop,timedig1,timedig2,stop,b);

//score table

score\_table scoreinsttop(clock,rst,logout\_pulse,green\_led\_user,internal\_id,auth\_bit,win,loose,disp\_button,level\_num,disp\_out,green\_max);

//number flashing seven segment

seven\_seg5 sevenseg\_inst(flash\_num, flash\_seg);

//level number seven segment

seven\_seg sevenseg\_level\_inst(level\_num,level\_seg);

//answer showing seven segment

seven\_seg sevenseg\_answer\_inst(seg\_in\_ans,seg\_answer);

//score showing seven segment

seven\_seg sevenseg\_score\_inst(disp\_out[7:4],seg\_score\_high);

seven\_seg sevenseg\_score\_inst2(disp\_out[3:0],seg\_score\_low);

//timer seven segment

seven\_seg sevenseg\_timer\_inst1(timedig2,time\_show\_high);

seven\_seg sevenseg\_timer\_inst2(timedig1,time\_show\_low);

endmodule