Memory Games:

Feature:

* **2 register**
  + 8 + 5 = 13 led (5 Note LEd + 8 bits LED bar for counting down)
    - LED BAR + connect to arrow side
    - BLUE LED connect positive to longer pin
  + 1 RED LED ( need 2 pin): RECORDING
* **Two player (OR single player)**
  + **Level(5) , win make it harder**
  + 11 button
    - 5 for each
    - 1 for start single player or restart
* LCD
  + **Press correct order**
  + **Display winner**
  + **logo(animation for win and lose)**
* Speaker
  + Play the key
  + Bee! When correct
* **Touch sensor**
  + **Start two player game**
  + **Recording input**

Pin Position

**ATMEGA1284:**

1(PB0)

2(PB1)

3(PB2)

4(PB3) passive speaker, -

5(PB4) SS RCLCK(LATCH)12, ATMEL

6(PB5) MOSI SER(DATA)14, ATMEL

7(PB6) ATMEL

8(PB7)SCK SRCLK(CLOCK)11

9(RESET) ATMEL

10(VCC) +, CAPACITOR1, ATMEL

11(GND) -, CAPACITOR1, ATMEL

PLAYER2

14(PD0) BUTTON6

15(PD1) BUTTON7

16(PD2) BUTTON8

17(PD3) BUTTON9

18(PD4) BUTTON10

20(PD6) LCD(4) (CONTROL)(RS)

21(PD7) LCD(6)(CONTROL)(E)

22(PC0) LCD(7)

23(PC1) LCD(8)

24(PC2) LCD(9)

25(PC3) LCD(10)

26(PC4) LCD(11)

27(PC5) LCD(12)

28(PC6) LCD(13)

29(PC7) LCD(14)

30(VCC) +, CAPACITOR2

31(GND) -, CAPACITOR2

TOUCH SENSOR

33(PA7) SIG

RECORDING LED

34(PA6) RED LED

RESET BUTTON/TWO PLAYER

35(PA5) BUTTON11

PLAYER1

36(PA4) BUTTON5

37(PA3) BUTTON4

38(PA2) BUTTON3

39(PA1) BUTTON2

40(PA0) BUTTON1

**LCD:**

1 -

2 +

3 potentiometer,-

4 D6(20)

5 -

6 D7(21)

7 C0(22)

8 C1(23)

9 C2(24)

10 C3(25)

11 C4(26)

12 C5(27)

13 C6(28)

14 C7(29)

15 +

16 -

**REGISTOR1:**

9 REGISTER2(14)

10 +

11 PB7(8)

12 PB4(5)

13 USE TO DISCONNECT RESGISTER2 ??

14 PB5(6)

**REGISTER2:**

1 BAR2

2 BAR3

3 BAR4

4 BAR5

5 BAR6

6 BAR7

7 BAR8

10 +

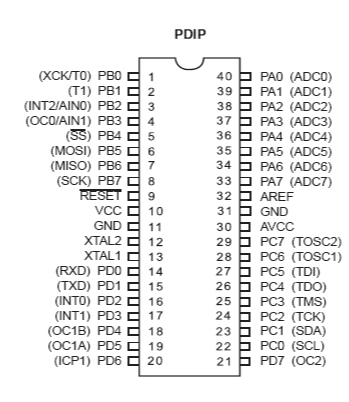
11 REGISTER1(11)

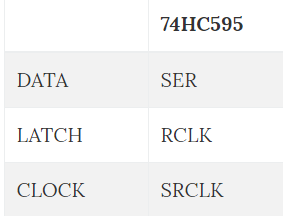
12 REGISTER1(12)

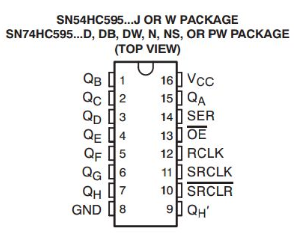
13 -

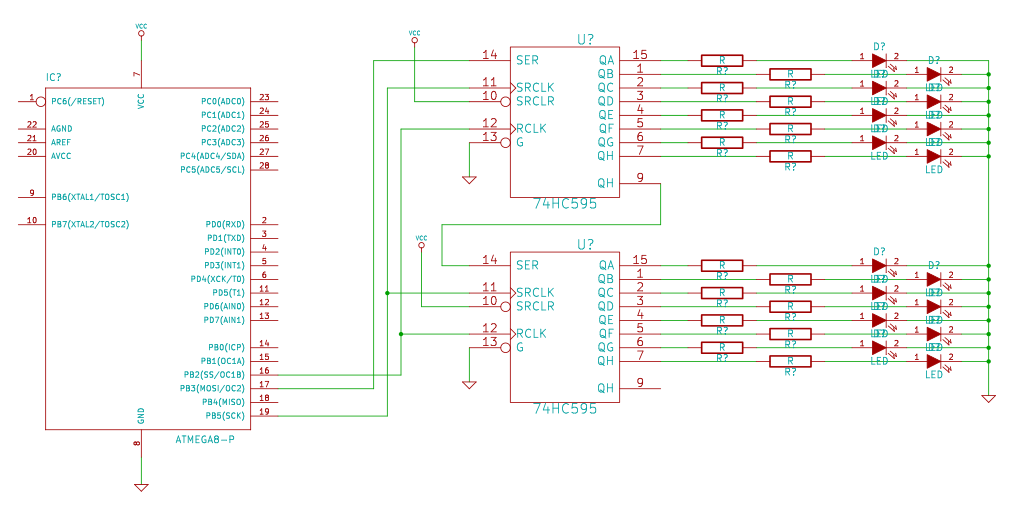
14 REGISTER1(9)

15 BAR1









**Note**

DDRA = 0x00; PORTA = 0xFF; // Configure PORTA as input, initialize to 1s

DDRB = 0xFF; PORTB = 0x00; // Configure PORTB as outputs, initialize to 0s

**Register code**

#include <avr/io.h>

#define SHIFT\_REGISTER DDRB

#define SHIFT\_PORT PORTB

#define BUTTON\_INPUT DDRA

#define BUTTON\_PORT PORTA

#define DATA (1 << PB5) //MOSI (SI)

#define LATCH (1 << PB4) //SS (RCK)

#define CLOCK (1 << PB7) //SCK (SCK)

#define BUTTON1 (1 << PA0)

#define LED ( 1 << PB0)

unsigned char button = 0x00;

void register\_input()

{

SHIFT\_REGISTER |= (DATA | LATCH | CLOCK ); //Set control pins as outputs

SHIFT\_PORT &= ~(DATA | LATCH | CLOCK); //Set control pins low

//Setup SPI

SPCR = (1<<SPE) | (1<<MSTR); //Start SPI as Master

//Pull LATCH low (Important: this is necessary to start the SPI transfer!)

SHIFT\_PORT &= ~LATCH;

//Shift in some data

// SPDR = 0b01010101; //This should light alternating LEDs

SPDR = 0xAA;

//Wait for SPI process to finish

while(!(SPSR & (1<<SPIF)));

//Shift in some more data since I have two shift registers hooked up

// SPDR = 0b01010101; //This should light alternating LEDs

//Wait for SPI process to finish

while(!(SPSR & (1<<SPIF)));

//Toggle latch to copy data to the storage register

SHIFT\_PORT |= LATCH;

SHIFT\_PORT &= ~LATCH;

}

int main(void)

{

//Setup IO

register\_input();

// BUTTON\_INPUT |= ~BUTTON1; //Set input

// BUTTON\_PORT = BUTTON1; //SET high

while(1)

{

}

}

**Speaker code**

/\* Partner(s) Name & E-mail: Jeff Trang (jtran094@ucr.edu), Kevin Tsai(ktsai017@ucr.edu), Xiangyu Chang(3750627@qq.com)

\* Lab Section: 022

\* Assignment: Lab #9 Exercise #3

\* Exercise Description: play a melody for a button press

\*

\* I acknowledge all content contained herein, excluding template or example

\* code, is my own original work.

\*/

#include <avr/io.h>

#include "timer.h"

// 0.954 hz is lowest frequency possible with this function,

// based on settings in PWM\_on()

// Passing in 0 as the frequency will stop the speaker from generating sound

void set\_PWM(double frequency) {

static double current\_frequency; // Keeps track of the currently set frequency

// Will only update the registers when the frequency changes, otherwise allows

// music to play uninterrupted.

if (frequency != current\_frequency) {

if (!frequency) { TCCR0B &= 0x08; } //stops timer/counter

else { TCCR0B |= 0x03; } // resumes/continues timer/counter

// prevents OCR3A from overflowing, using prescaler 64

// 0.954 is smallest frequency that will not result in overflow

if (frequency < 0.954) { OCR0A = 0xFFFF; }

// prevents OCR0A from underflowing, using prescaler 64 // 31250 is largest frequency that will not result in underflow

else if (frequency > 31250) { OCR0A = 0x0000; }

// set OCR3A based on desired frequency

else { OCR0A = (short)(8000000 / (128 \* frequency)) - 1; }

TCNT0 = 0; // resets counter

current\_frequency = frequency; // Updates the current frequency

}

}

void PWM\_on() {

TCCR0A = (1 << COM0A0) | (1 << WGM00);

// COM3A0: Toggle PB3 on compare match between counter and OCR0A

TCCR0B = (1 << WGM02) | (1 << CS01) | (1 << CS00);

// WGM02: When counter (TCNT0) matches OCR0A, reset counter

// CS01 & CS30: Set a prescaler of 64

set\_PWM(0);

}

void PWM\_off() {

TCCR0A = 0x00;

TCCR0B = 0x00;

}

unsigned char tmpA;

enum states {start, on, off, hold} state;

//double array[8] = {C261.63, D293.66, E329.63, F349.23, G392.00, A440.00, B493.88, C523.25}; //C4-C5

double array[13] = {440.00, 493.88, 523.25, 493.88, 440.00, 493.88, 392.00, 440.00, 392.00, 349.23, 392.00, 349.23, 329.63};

//a b c b a b g

int i = 0; //count the timer pulse

int count = 0; //index of melody

void Tick()

{

switch(state) //transition

{

case start:

state = off;

break;

case off:

if (tmpA == 0x01) // on

{

i = 0; //reset

count =0;

state = on;

}

break;

case on:

if (i == 30 ) //when melody end

{

state = hold; //hold state

}

break;

case hold:

if (tmpA == 0x00) //if release button

{

state = off;

}

break;

default:

break;

}

switch(state) //action

{

case start:

break;

case off:

set\_PWM(0.00); //turn off

break;

case on:

if (i == 0)

{

set\_PWM(array[count]);

count++;

}

else if (i== 1)

{

set\_PWM(array[count]);

count++;

}

else if (i== 2)

{

set\_PWM(array[count]);

count++;

}

else if (i== 4)

{

set\_PWM(array[count]);

count++;

}

else if (i== 5)

{

set\_PWM(array[count]);

count++;

}

else if (i == 9)

{

set\_PWM(array[count]);

count++;

}

else if (i == 10)

{

set\_PWM(array[count]);

count++;

}

else if (i == 13) //special pulse, no sound

{

set\_PWM(0.00);

}

else if (i == 16)

{

set\_PWM(array[count]);

count++;

}

else if (i == 17)

{

set\_PWM(array[count]);

count++;

}

else if (i == 18)

{

set\_PWM(array[count]);

count++;

}

else if (i == 21)

{

set\_PWM(array[count]);

count++;

}

else if (i == 22)

{

set\_PWM(array[count]);

count++;

}

else if (i == 26)

{

set\_PWM(array[count]);

count++;

}

i++;

break;

case hold:

set\_PWM(0.00);

break;

default:

break;

}

}

int main() {

DDRA = 0x00; PORTA = 0xFF;//input, initial at 1

DDRB = 0xFF; PORTB = 0x00;//output, initial at 0

PWM\_on();

TimerSet(200);

TimerOn();

state = start;

while(1) {

tmpA = ~PINA;//make sure only PA1-PA0 work

Tick();

while (!TimerFlag){} // Wait for BL's period

TimerFlag = 0; // Lower flag

}

}

**Final code**

#include <avr/io.h>

#define SHIFT\_REGISTER DDRB

#define SHIFT\_PORT PORTB

#define DATA (1 << PB5) //MOSI (SI)

#define LATCH (1 << PB4) //SS (RCK)

#define CLOCK (1 << PB7) //SCK (SCK)

unsigned char tempA = 0x00;

unsigned char tempD = 0x00;

unsigned char tempLED = 0x00;

void register\_setup()

{

SHIFT\_REGISTER |= (DATA | LATCH | CLOCK ); //Set control pins as outputs

SHIFT\_PORT &= ~(DATA | LATCH | CLOCK); //Set control pins low

//Setup SPI

SPCR = (1<<SPE) | (1<<MSTR); //Start SPI as Master

//Pull LATCH low (Important: this is necessary to start the SPI transfer!)

SHIFT\_PORT &= ~LATCH;

//Shift in some data

// SPDR = 0b01010101; //This should light alternating LEDs

SPDR = 0xFF;

//Wait for SPI process to finish

while(!(SPSR & (1<<SPIF)));

//Shift in some more data since I have two shift registers hooked up

SPDR = 0x00; //This should light alternating LEDs

//Wait for SPI process to finish

while(!(SPSR & (1<<SPIF)));

//Toggle latch to copy data to the storage register

SHIFT\_PORT |= LATCH;

SHIFT\_PORT &= ~LATCH;

}

void set\_register()

{

//Shift in some data

SPDR =0x00; //This should light alternating LEDs

//SPDR = 0xAA;

//Wait for SPI process to finish

while(!(SPSR & (1<<SPIF)));

//Shift in some more data since I have two shift registers hooked up

SPDR = tempLED; //This should light alternating LEDs

//Wait for SPI process to finish

while(!(SPSR & (1<<SPIF)));

//Toggle latch to copy data to the storage register

SHIFT\_PORT |= LATCH;

SHIFT\_PORT &= ~LATCH;

}

enum led\_states {led\_init, wait, hold} led\_state;

void set\_led()

{

switch(led\_state) //combine action and transition

{

case led\_init:

led\_state = wait;

break;

case wait:

if (tempA == 0x01 || tempD == 0x01)

{

tempLED = 0x01;

led\_state = hold;

}

else if (tempA == 0x02 || tempD == 0x02)

{

tempLED = 0x02;

led\_state = hold;

}

else if(tempA == 0x04 || tempD == 0x04)

{

tempLED = 0x04;

led\_state = hold;

}

else if(tempA == 0x08 || tempD == 0x08)

{

tempLED = 0x08;

led\_state = hold;

}

else if(tempA == 0x10 || tempD == 0x10)

{

tempLED = 0x10;

led\_state = hold;

}

break;

case hold:

if ( (tempA == 0x00) && (tempD == 0x00))

{

tempLED = 0x00;

led\_state = wait;

}

break;

default:

break;

}

}

int main(void)

{

//Setup IO

DDRD = 0xC0; PORTD = 0x3F; //for lcd D7D6, D4-D0 for button

DDRA = 0x00;

PORTA = 0xFF;

register\_setup();

//light\_state = light\_init;

//led\_state = led\_init;

while(1)

{

//Loop forever

tempA = ~PINA;

tempD = ~PIND & 0x1F;

set\_led();

set\_register();

}

}

======================================================================

//work with sequence, double tap sensor, lose, reset, speaker, winnner music

#include <avr/io.h>

#include "timer.h"

#include "io.c"

#include "bit.h"

#define SHIFT\_REGISTER DDRB

#define SHIFT\_PORT PORTB

#define DATA (1 << PB5) //MOSI (SI)

#define LATCH (1 << PB4) //SS (RCK)

#define CLOCK (1 << PB7) //SCK (SCK)

#define RED\_LED (1 << PB0) //red led

#define TOUCH ( 1 << PA5) //touch sensor

unsigned char tempA = 0x00; //PINA

unsigned char tempD = 0x00; // PINB

unsigned char tempLED = 0x00; //send to led

int record = 0; //check if it is recording

unsigned char press = 0x00; //what button was press

unsigned char start\_game = 0x00;

unsigned char mark = 0x00; //when a button is press and release

int counter = 0; //check notes in the level

int play\_counter = 0; //counting for playing note

int play\_level = 0 ;// count how many note in the melody

unsigned char level = 1;

unsigned char win = 0;

unsigned char lose = 0;

unsigned char play = 0;

unsigned char level1[3] = {0x01, 0x04,0x02} ;// 3 note 1 3 2

unsigned char level2[5] = {0x08, 0x01, 0x08, 0x10, 0x02}; // 5 note 4 1 4 5 2

unsigned char level3[7] = {0x02, 0x10, 0x08, 0x10, 0x01, 0x04, 0x02};// 7 note 2 5 4 5 1 3 2

unsigned char level4[5] = {0x08, 0x04, 0x01, 0x02, 0x01};// 5 note, faster 4 3 1 2 1

unsigned char level5[7] = {0x01, 0x10, 0x04, 0x02, 0x08, 0x02, 0x01};// 7 note faster 1 5 3 2 4 2 1

double correct[2] = {1174.659, 987.767};

double winner[10] = {329.628, 0.00,329.628,0.00,329.628,261.626, 293.665, 329.628, 293.665, 329.628};

unsigned long change\_time = 10;

int speaker\_count= 0;

unsigned char game\_start = 0;

int k =0;

int play\_sound; //don't interrupt

//light up led in game

void reset(); //prototype

void speaker\_counter();

void win\_sound();

/////////////////speaker function//////////////////////////////////////////////////////////////////

// 0.954 hz is lowest frequency possible with this function,

// based on settings in PWM\_on()

// Passing in 0 as the frequency will stop the speaker from generating sound

void set\_PWM(double frequency) {

static double current\_frequency; // Keeps track of the currently set frequency

// Will only update the registers when the frequency changes, otherwise allows

// music to play uninterrupted.

if (frequency != current\_frequency) {

if (!frequency) { TCCR0B &= 0x08; } //stops timer/counter

else { TCCR0B |= 0x03; } // resumes/continues timer/counter

// prevents OCR3A from overflowing, using prescaler 64

// 0.954 is smallest frequency that will not result in overflow

if (frequency < 0.954) { OCR0A = 0xFFFF; }

// prevents OCR0A from underflowing, using prescaler 64 // 31250 is largest frequency that will not result in underflow

else if (frequency > 31250) { OCR0A = 0x0000; }

// set OCR3A based on desired frequency

else { OCR0A = (short)(8000000 / (128 \* frequency)) - 1; }

TCNT0 = 0; // resets counter

current\_frequency = frequency; // Updates the current frequency

}

}

void PWM\_on() {

TCCR0A = (1 << COM0A0) | (1 << WGM00);

// COM3A0: Toggle PB3 on compare match between counter and OCR0A

TCCR0B = (1 << WGM02) | (1 << CS01) | (1 << CS00);

// WGM02: When counter (TCNT0) matches OCR0A, reset counter

// CS01 & CS30: Set a prescaler of 64

set\_PWM(0);

}

void PWM\_off() {

TCCR0A = 0x00;

TCCR0B = 0x00;

}

////////////////speaker function end//////////////////////////////////////////////////////////////

//display/change in game led speed

enum game\_led\_states {game\_init, game\_off, game\_on} game\_led\_state ;

void game\_led()

{

switch(game\_led\_state)

{

case game\_init:

game\_led\_state = game\_off;

break;

case game\_off:

if (play == 1)

{

game\_led\_state = game\_on;

}

break;

case game\_on:

if (play\_counter == play\_level)

{

game\_led\_state = game\_off;

play\_counter = 0; //reset note counter

play = 0; //stop playing

tempLED = 0x00;

}

break;

default:

break;

}

switch(game\_led\_state)

{

case game\_init:

break;

case game\_off:

break;

case game\_on:

if (level == 1)

{

change\_time = 10; //keep same speed

tempLED = level1[play\_counter];

play\_counter++;

}

else if (level == 2)

{

change\_time = 10; //keep same speed

tempLED = level2[play\_counter];

play\_counter++;

}

else if (level == 3)

{

change\_time = 10; //keep same speed

tempLED = level3[play\_counter];

play\_counter++;

}

else if (level == 4)

{

change\_time = 8; //change speed

tempLED = level4[play\_counter];

play\_counter++;

}

else if (level == 5)

{

change\_time = 8; //change speed

tempLED = level5[play\_counter];

play\_counter++;

}

break;

default:

break;

}

}

//game level up

enum level\_states {level\_init, levelone, leveltwo, levelthree, levelfour, levelfive} level\_state;

void game\_level()

{

switch(level\_state) //transition

{

case level\_init:

if(level == 1 && start\_game == 1)

{

level =4;

play = 1;

level\_state = levelfour; //testing

play\_level = 5;

//level\_state = levelone;

//play\_level = 3;

start\_game = 0;

}

break;

case levelone:

if (level == 2)

{

play = 1;

level\_state = leveltwo;

play\_level = 5;

}

break;

case leveltwo:

if (level == 3)

{

play = 1;

level\_state = levelthree;

play\_level = 7;

}

break;

case levelthree:

if (level == 4)

{

play = 1;

level\_state = levelfour;

play\_level = 5;

}

break;

case levelfour:

if (level == 5)

{

play = 1;

level\_state = levelfive;

play\_level = 7;

}

break;

case levelfive:

break;

default:

break;

}

switch(level\_state) //action

{

case level\_init:

break;

case levelone:

if (mark == 1 && GetBit(PORTB, 0)==1 )

{

mark = 0;

if (level1[counter] == press)

{

counter+=1;

if (counter == 3)

{

level = 2;

counter = 0; //reset counter

PORTB = SetBit(PORTB,0, 0 ); //set PB0 to 0

}

}

else if (press != 0x00)

{

lose = 1;

reset();

// PORTB = SetBit(PORTB, 1,1 );

}

}

break;

case leveltwo:

if (mark == 1 && GetBit(PORTB, 0)==1)

{

mark = 0;

if (level2[counter] == press)

{

counter+=1;

if (counter == 5)

{

level = 3;

counter = 0;

PORTB = SetBit(PORTB,0, 0 ); //set PB0 to 0

}

}

else if (press != 0x00)

{

lose = 1;

reset();

}

}

break;

case levelthree:

if (mark == 1 && GetBit(PORTB, 0)==1 )

{

mark = 0;

if (level3[counter] == press)

{

counter+=1;

if (counter == 7)

{

level = 4;

counter = 0;

PORTB = SetBit(PORTB,0, 0 ); //set PB0 to 0

}

}

else if (press != 0x00)

{

lose = 1;

reset();

}

}

break;

case levelfour:

if (mark == 1 && GetBit(PORTB, 0)==1 )

{

mark = 0;

if (level4[counter] == press)

{

counter+=1;

if (counter == 5)

{

level = 5;

counter = 0; //reset counter

PORTB = SetBit(PORTB,0, 0 ); //set PB0 to 0

}

}

else if (press != 0x00)

{

lose = 1;

reset();

}

}

break;

case levelfive:

if (mark == 1 && GetBit(PORTB, 0)==1 )

{

mark = 0;

if (level5[counter] == press)

{

counter+=1;

if (counter == 7)

{

win = 1;

PORTB = SetBit(PORTB,0, 0 ); //set PB0 to 0

//win\_sound();

}

}

else if (press != 0x00)

{

lose = 1;

reset();

}

}

break;

default:

break;

}

}

void testing()

{

if (k == 0)

{

tempLED = level1[0];

}

else if (k == 1)

{

tempLED = level1[1];

}

else if ( k == 2)

{

tempLED = level1[2];

}

k++;

}

//set up register

void register\_setup()

{

SHIFT\_REGISTER |= (DATA | LATCH | CLOCK | RED\_LED | 0x02 | 0x04 | 0x08); //Set control pins as outputs, and red led

SHIFT\_PORT &= ~(DATA | LATCH | CLOCK | RED\_LED | 0x02 | 0x04 | 0x08); //Set control pins low, and red led

//Setup SPI

SPCR = (1<<SPE) | (1<<MSTR); //Start SPI as Master

//Pull LATCH low (Important: this is necessary to start the SPI transfer!)

SHIFT\_PORT &= ~LATCH;

//Shift in some data

// SPDR = 0b01010101; //This should light alternating LEDs

SPDR = 0xFF;

//Wait for SPI process to finish

while(!(SPSR & (1<<SPIF)));

//Shift in some more data since I have two shift registers hooked up

SPDR = 0x00; //This should light alternating LEDs

//Wait for SPI process to finish

while(!(SPSR & (1<<SPIF)));

//Toggle latch to copy data to the storage register

SHIFT\_PORT |= LATCH;

SHIFT\_PORT &= ~LATCH;

}

//change led in the register

void set\_register()

{

//Shift in some data

SPDR =0xFF; //This should light alternating LEDs

//SPDR = 0xAA;

//Wait for SPI process to finish

while(!(SPSR & (1<<SPIF)));

//Shift in some more data since I have two shift registers hooked up

SPDR = tempLED; //This should light alternating LEDs

speaker\_counter();

//Wait for SPI process to finish

while(!(SPSR & (1<<SPIF)));

//Toggle latch to copy data to the storage register

SHIFT\_PORT |= LATCH;

SHIFT\_PORT &= ~LATCH;

}

enum led\_states {led\_init, wait, hold} led\_state;

void set\_led()

{

tempA &= ~TOUCH; //ignore touch snesor

switch(led\_state) //combine action and transition

{

case led\_init:

led\_state = wait;

break;

case wait:

if (tempA == 0x01 || tempD == 0x01)

{

tempLED = 0x01;

led\_state = hold;

}

else if (tempA == 0x02 || tempD == 0x02)

{

tempLED = 0x02;

led\_state = hold;

}

else if(tempA == 0x04 || tempD == 0x04)

{

tempLED = 0x04;

led\_state = hold;

}

else if(tempA == 0x08 || tempD == 0x08)

{

tempLED = 0x08;

led\_state = hold;

}

else if(tempA == 0x10 || tempD == 0x10)

{

tempLED = 0x10;

led\_state = hold;

}

else if (tempA == 0x40)

{

level = 1;

game\_led\_state = game\_init;

level\_state = level\_init;

start\_game = 1;

play = 1;

}

break;

case hold:

if ( (tempA == 0x00) && (tempD == 0x00))

{

mark = 1;

press = tempLED;

tempLED = 0x00;

led\_state = wait;

}

break;

default:

break;

}

}

enum sensor\_states {sensor\_init, off, on} sensor\_state;

void set\_sensor()

{

switch(sensor\_state)

{

case sensor\_init:

sensor\_state = off;

break;

case off:

if (GetBit(PINA, 5) ) //if sensor is on

{

sensor\_state = on;

}

break;

case on:

if (~GetBit(PINA, 5)) //if sensor is off

{

sensor\_state = off;

}

break;

default:

break;

}

switch(sensor\_state)

{

case sensor\_init:

break;

case off:

//PORTB = SetBit(PORTB,0, 0 ); //set PB0 to 0

record = 0;

break;

case on:

PORTB = SetBit(PORTB,0, 1 ); //set PB0 to 1

record = 1;

press = 0x00;

break;

default:

break;

}

}

//lose or win

void reset()

{

if (lose == 1 )

{

PORTB = SetBit(PORTB, 2,1);

tempLED = 0x00; //send to led

record = 0; //check if it is recording

press = 0x00; //what button was press

start\_game = 0x00;

mark = 0x00; //when a button is press and release

counter = 0; //check notes in the level

play\_counter = 0; //counting for playing note

play\_level = 0 ;// count how many note in the melody

level = 1;

win = 0;

lose = 0;

play = 0;

led\_state = led\_init;

sensor\_state = sensor\_init;

game\_led\_state = game\_init;

level\_state = level\_init;

PORTB = SetBit(PORTB, 0, 0); //reset recording button

}

}

//change hex into value, and play

void speaker\_counter()

{

if (tempLED == 0x01)

{

set\_PWM(261.63);

}

else if (tempLED == 0x02)

{

set\_PWM(293.66);

}

else if (tempLED == 0x04)

{

set\_PWM(329.63);

}

else if (tempLED == 0x08)

{

set\_PWM(349.23);

}

else if (tempLED == 0x10)

{

set\_PWM(392.00);

}

else if (tempLED == 0x00 && win ==0)

{

set\_PWM(0.00);

}

}

void win\_sound()

{

if (win == 1)

{

if (speaker\_count== 10 )

{

win = 0;

}

else

{

PORTB = SetBit(PORTB, 1, 1);

set\_PWM(winner[speaker\_count]);

speaker\_count++;

}

}

}

int main(void)

{

//need double tap of sensor to make it work

//Setup IO

DDRD = 0xC0; PORTD = 0x3F; //for lcd D7D6, D4-D0 for button

DDRA = 0x00; PORTA = 0xFF; //A7 for output

unsigned long game\_led\_elapsed = 0;

PWM\_on();

TimerSet(50);

TimerOn();

register\_setup();

//unsigned long period = 30000;

unsigned char win\_elapsed = 0;

led\_state = led\_init;

sensor\_state = sensor\_init;

game\_led\_state = game\_init;

level\_state = level\_init;

game\_level(); //initialize the game first

lose = 0;

win = 0;

while(1)

{

//PORTB |= RED\_LED; // MAKE red RECORD SHINE

//Loop forever

tempA = ~PINA;

tempD = ~PIND & 0x1F;

if (win\_elapsed >=10)

{

win\_sound();

win\_elapsed = 0;

}

set\_led();

set\_register();

set\_sensor();

game\_level();

if ( game\_led\_elapsed >= change\_time)

{

game\_led();

game\_led\_elapsed =0 ;

}

while (!TimerFlag){} // Wait for BL's period

TimerFlag = 0; // Lower flag

game\_led\_elapsed += 1;

win\_elapsed +=1;

//set\_record();

}

}