/\*\*

\* 6-19-2007

\* Copyright 2009, Spark Fun Electronics

\* Nathan Seidle

\* nathan at sparkfun.com

\*

\* Released under the Creative Commons Attribution Share-Alike 3.0 License

\* http://creativecommons.org/licenses/by-sa/3.0

\*

\* Simon Game ported for the ATmega328

\* Fixes and cleanup by Joshua Neal <joshua[at]trochotron.com>

\*

\* Generates random sequence, plays music, and displays button lights.

\*

\* Simon tones from Wikipedia

\* - A (red, upper left) - 440Hz - 2.272ms - 1.136ms pulse

\* - a (green, upper right, an octave higher than A) - 880Hz - 1.136ms,

\* 0.568ms pulse

\* - D (blue, lower left, a perfect fourth higher than the upper left)

\* 587.33Hz - 1.702ms - 0.851ms pulse

\* G (yellow, lower right, a perfect fourth higher than the lower left) -

\* 784Hz - 1.276ms - 0.638ms pulse

\* Select the - Lilypad for arduino - board, when programming with ftdi connector

\* Power up with AA's and turn on - then program it.

\* The tones are close, but probably off a bit, but they sound all right.

\* The current version of Simon uses the ATmega328. The external osciallator

\* was removed to reduce component count. This version of simon relies on the

\* internal default 1MHz osciallator. Do not set the external fuses.

\*

\* Modified October 2013 by Stephen O'Gara to add solenoid activation upon winning the game.

\* Lengthened the winner sounds and lights series.

\*/

#include <avr/io.h>

#include <avr/interrupt.h>

#define BOARD\_REV\_PTH

#ifdef BOARD\_REV\_PTH

#define CHIP\_ATMEGA168

#define LED\_RED (1 << 0)

#define LED\_GREEN (1 << 1)

#define LED\_BLUE (1 << 2)

#define LED\_YELLOW (1 << 3)

/\* LED pin definitions \*/

#define LED\_RED\_PIN 2

#define LED\_RED\_PORT PORTB

#define LED\_GREEN\_PIN 3

#define LED\_GREEN\_PORT PORTD

#define LED\_BLUE\_PIN 5

#define LED\_BLUE\_PORT PORTB

#define LED\_YELLOW\_PIN 5

#define LED\_YELLOW\_PORT PORTD

/\*solenoid pin definition\*/

int solenoid1 = A0; // solenoid1 connected to analog pin A0

/\* Button pin definitions \*/

#define BUTTON\_RED\_PIN 1

#define BUTTON\_RED\_PORT PINB

#define BUTTON\_GREEN\_PIN 2

#define BUTTON\_GREEN\_PORT PIND

#define BUTTON\_BLUE\_PIN 4

#define BUTTON\_BLUE\_PORT PINB

#define BUTTON\_YELLOW\_PIN 6

#define BUTTON\_YELLOW\_PORT PIND

/\* Buzzer pin definitions \*/

#define BUZZER1 4

#define BUZZER1\_PORT PORTD

#define BUZZER2 7

#define BUZZER2\_PORT PORTD

#endif /\* BOARD\_REV\_PTH \*/

/\* Define game parameters \*/

/\* Change this number to around 12 when ready for final adjustments \*/

#define MOVES\_TO\_WIN 8

#define TIME\_LIMIT 3000 /\* 3000ms = 3 sec \*/

#define sbi(port\_name, pin\_number) (port\_name |= 1<<pin\_number)

#define cbi(port\_name, pin\_number) \

((port\_name) &= (uint8\_t)~(1 << pin\_number))

/\* Declarations for static functions \*/

void delay\_us(uint16\_t delay);

void delay\_ms(uint16\_t delay);

uint8\_t check\_button(void);

void set\_leds(uint8\_t leds);

void buzz\_sound(uint16\_t buzz\_length\_ms, uint16\_t buzz\_delay);

void toner(uint8\_t tone, uint16\_t buzz\_length\_ms);

void add\_to\_moves(void);

void play\_moves(void);

void play\_loser(void);

void play\_winner(void);

void ioinit(void);

/\* Game state \*/

uint8\_t moves[32];

uint8\_t nmoves = 0;

ISR (SIG\_OVERFLOW2)

{

/\*

\* Prescalar of 1024

\* Clock = 16MHz

\* 15,625 clicks per second

\* 64us per click

\*/

/\* Preload timer 2 for 125 clicks. Should be 8ms per ISR call \*/

TCNT2 = 131; /\* 256 - 125 = 131 \*/

}

/\* General short delays, using internal timer do a fairly accurate 1us \*/

#ifdef CHIP\_ATMEGA168

void

delay\_us(uint16\_t delay)

{

while (delay > 256)

{

TIFR0 = (1<<TOV0); /\* Clear any interrupt flags on Timer0 \*/

TCNT0 = 0;

while ( (TIFR0 & (1<<TOV0)) == 0);

delay -= 256;

}

TIFR0 = (1<<TOV0); /\* Clear any interrupt flags on Timer0 \*/

/\*

\* 256 - 125 = 131 : Preload timer 0 for x clicks.

\* Should be 1us per click

\*/

TCNT0 = 256 - delay;

while ((TIFR0 & (1<<TOV0)) == 0) {

/\* do nothing \*/

}

}

#endif

/\* General short delays \*/

void

delay\_ms(uint16\_t x)

{

while (x-- > 0) {

delay\_us(1000);

}

}

/\* Light the given set of LEDs \*/

void

set\_leds(uint8\_t leds)

{

if ((leds & LED\_RED) != 0) {

sbi(LED\_RED\_PORT, LED\_RED\_PIN);

}

else {

cbi(LED\_RED\_PORT, LED\_RED\_PIN);

}

if ((leds & LED\_GREEN) != 0) {

sbi(LED\_GREEN\_PORT, LED\_GREEN\_PIN);

}

else {

cbi(LED\_GREEN\_PORT, LED\_GREEN\_PIN);

}

if ((leds & LED\_BLUE) != 0) {

sbi(LED\_BLUE\_PORT, LED\_BLUE\_PIN);

}

else {

cbi(LED\_BLUE\_PORT, LED\_BLUE\_PIN);

}

if ((leds & LED\_YELLOW) != 0) {

sbi(LED\_YELLOW\_PORT, LED\_YELLOW\_PIN);

}

else {

cbi(LED\_YELLOW\_PORT, LED\_YELLOW\_PIN);

}

}

#ifdef BOARD\_REV\_PTH

void

init\_gpio(void)

{

/\* 1 = output, 0 = input \*/

DDRB = 0b11101101; /\* LEDs and Buttons \*/

DDRC = 0b11111111; /\* LEDs and Buttons \*/

DDRD = 0b10111011; /\* LEDs, buttons, buzzer, TX/RX \*/

PORTB = 0b00010010; /\* Enable pull-ups on buttons 1,4 \*/

//PORTC = 0b00100110; /\* Enable pull-ups on buttons 0,2,3 \*/

PORTD = 0b01000100; /\* Enable pull-up on button 1 \*/

}

#endif

void

ioinit(void)

{

init\_gpio();

//Set Timer 0 Registers to Default Setting to over-ride the timer initialization made in the init() function of the Arduino Wiring libary (Wiring.c in the hardware/core/arduino folder)

TCCR0A = 0;

TIMSK0 = 0;

/\* Init timer 0 for delay\_us timing (1,000,000 / 1 = 1,000,000) \*/

//TCCR0B = (1<<CS00); /\* Set Prescaler to 1. CS00=1 \*/

TCCR0B = (1<<CS01); /\* Set Prescaler to 1. CS00=1 \*/

/\* Init timer 2 \*/

ASSR = 0;

/\* Set Prescaler to 1024. CS22=1, CS21=1,CS20=1 \*/

TCCR2B = (1<<CS22)|(1<<CS21)|(1<<CS20);

TIMSK2 = (1<<TOIE2); /\* Enable Timer 2 Interrupt \*/

cli(); //We don't use any interrupt functionality. Let's turn it off so Arduino doesn't screw around with it!

}

/\* Returns a '1' bit in the position corresponding to LED\_RED, etc. \*/

uint8\_t

check\_button(void)

{

uint8\_t button\_pressed = 0;

if ((BUTTON\_RED\_PORT & (1 << BUTTON\_RED\_PIN)) == 0)

button\_pressed |= LED\_RED;

if ((BUTTON\_GREEN\_PORT & (1 << BUTTON\_GREEN\_PIN)) == 0)

button\_pressed |= LED\_GREEN;

if ((BUTTON\_BLUE\_PORT & (1 << BUTTON\_BLUE\_PIN)) == 0)

button\_pressed |= LED\_BLUE;

if ((BUTTON\_YELLOW\_PORT & (1 << BUTTON\_YELLOW\_PIN)) == 0)

button\_pressed |= LED\_YELLOW;

return button\_pressed;

}

/\* Play the loser sound/lights \*/

void

play\_loser(void)

{

set\_leds(LED\_RED|LED\_GREEN);

buzz\_sound(255, 1500);

set\_leds(LED\_BLUE|LED\_YELLOW);

buzz\_sound(255, 1500);

set\_leds(LED\_RED|LED\_GREEN);

buzz\_sound(255, 1500);

set\_leds(LED\_BLUE|LED\_YELLOW);

buzz\_sound(255, 1500);

}

/\* Play the winner sound \*/

void

winner\_sound(void)

{

uint8\_t x, y;

/\* Toggle the buzzer at various speeds \*/

/\* changed loop up to 9 times for Y< , then changed it back \*/

for (x = 250; x > 70; x--) {

for (y = 0; y < 4; y++) {

sbi(BUZZER2\_PORT, BUZZER2);

cbi(BUZZER1\_PORT, BUZZER1);

delay\_us(x);

cbi(BUZZER2\_PORT, BUZZER2);

sbi(BUZZER1\_PORT, BUZZER1);

delay\_us(x);

}

}

}

/\* Play the winner sound and lights \*/

/\* added in the digital write lines for solenoid1 on pin A0\*/

void

play\_winner(void)

{

set\_leds(LED\_GREEN|LED\_BLUE);

winner\_sound();

set\_leds(LED\_RED|LED\_YELLOW);

winner\_sound();

set\_leds(LED\_GREEN|LED\_BLUE);

winner\_sound();

set\_leds(LED\_RED|LED\_YELLOW);

winner\_sound();

set\_leds(LED\_GREEN|LED\_BLUE);

winner\_sound();

set\_leds(LED\_RED|LED\_YELLOW);

winner\_sound();

set\_leds(LED\_GREEN|LED\_BLUE);

winner\_sound();

set\_leds(LED\_RED|LED\_YELLOW);

winner\_sound();

set\_leds(LED\_GREEN|LED\_BLUE);

winner\_sound();

set\_leds(LED\_RED|LED\_YELLOW);

winner\_sound();

set\_leds(LED\_GREEN|LED\_BLUE);

winner\_sound();

set\_leds(LED\_RED|LED\_YELLOW);

winner\_sound();

/\* Added in this solenoid output with a 2 second hold \*/

digitalWrite(solenoid1, HIGH); // sets the solenoid on

delay\_ms(2000); // waits for 9 seconds

digitalWrite(solenoid1, LOW); // sets the solenoid off

delay\_ms(3000); // waits for a second

/\* need to turn off the lights around here. Then it would be great if the sleep mode could be used to reduce power \*/

/\* Turn off all LEDs \*/

set\_leds(0);

}

/\* Plays the current contents of the game moves \*/

void

play\_moves(void)

{

uint8\_t move;

for (move = 0; move < nmoves; move++) {

toner(moves[move], 150);

delay\_ms(150);

}

}

/\* Adds a new random button to the game sequence, by sampling the timer \*/

void

add\_to\_moves(void)

{

uint8\_t new\_button;

/\* Use the lower 2 bits of the timer for the random value \*/

new\_button = 1 << (TCNT2 & 0x3);

moves[nmoves++] = new\_button;

}

/\* Toggle buzzer every buzz\_delay\_us, for a duration of buzz\_length\_ms. \*/

void

buzz\_sound(uint16\_t buzz\_length\_ms, uint16\_t buzz\_delay\_us)

{

uint32\_t buzz\_length\_us;

buzz\_length\_us = buzz\_length\_ms \* (uint32\_t)1000;

while (buzz\_length\_us > buzz\_delay\_us\*2) {

buzz\_length\_us -= buzz\_delay\_us\*2;

/\* toggle the buzzer at various speeds \*/

cbi(BUZZER1\_PORT, BUZZER1);

sbi(BUZZER2\_PORT, BUZZER2);

delay\_us(buzz\_delay\_us);

sbi(BUZZER1\_PORT, BUZZER1);

cbi(BUZZER2\_PORT, BUZZER2);

delay\_us(buzz\_delay\_us);

}

}

/\*

\* Light an LED and play tone

\*

\* red, upper left: 440Hz - 2.272ms - 1.136ms pulse

\* green, upper right: 880Hz - 1.136ms - 0.568ms pulse

\* blue, lower left: 587.33Hz - 1.702ms - 0.851ms pulse

\* yellow, lower right: 784Hz - 1.276ms - 0.638ms pulse

\*/

void

toner(uint8\_t which, uint16\_t buzz\_length\_ms)

{

set\_leds(which);

switch (which) {

case LED\_RED:

buzz\_sound(buzz\_length\_ms, 1136);

break;

case LED\_GREEN:

buzz\_sound(buzz\_length\_ms, 568);

break;

case LED\_BLUE:

buzz\_sound(buzz\_length\_ms, 851);

break;

case LED\_YELLOW:

buzz\_sound(buzz\_length\_ms, 638);

break;

}

/\* Turn off all LEDs \*/

set\_leds(0);

}

/\* Show an "attract mode" display while waiting for user to press button. \*/

void

attract\_mode(void)

{

while (1) {

set\_leds(LED\_RED);

delay\_ms(100);

if (check\_button() != 0x00)

return;

set\_leds(LED\_BLUE);

delay\_ms(100);

if (check\_button() != 0x00)

return;

set\_leds(LED\_GREEN);

delay\_ms(100);

if (check\_button() != 0x00)

return;

set\_leds(LED\_YELLOW);

delay\_ms(100);

if (check\_button() != 0x00)

return;

}

}

/\* Wait for a button to be pressed. Returns one of led colors (LED\_RED, etc.)

\* if successful, 0 if timed out \*/

uint8\_t

wait\_for\_button(void)

{

uint16\_t time\_limit = TIME\_LIMIT;

uint8\_t released = 0;

uint8\_t old\_button;

while (time\_limit > 0) {

uint8\_t button;

/\* Implement a small bit of debouncing \*/

old\_button = button;

button = check\_button();

/\*

\* Make sure we've seen the previous button

\* released before accepting new buttons

\*/

if (button == 0)

released = 1;

if (button == old\_button && released == 1) {

/\* Make sure just one button is pressed \*/

if (button == LED\_RED ||

button == LED\_BLUE ||

button == LED\_GREEN ||

button == LED\_YELLOW) {

return button;

}

}

delay\_ms(1);

time\_limit--;

}

return 0; /\* Timed out \*/

}

/\* Play the game. Returns 0 if player loses, or 1 if player wins. \*/

int

game\_mode(void)

{

nmoves = 0;

while (nmoves < MOVES\_TO\_WIN) {

uint8\_t move;

/\* Add a button to the current moves, then play them back \*/

add\_to\_moves();

play\_moves();

/\* Then require the player to repeat the sequence. \*/

for (move = 0; move < nmoves; move++) {

uint8\_t choice = wait\_for\_button();

/\* If wait timed out, player loses. \*/

if (choice == 0)

return 0;

toner(choice, 150);

/\* If the choice is incorect, player loses. \*/

if (choice != moves[move]) {

return 0;

}

}

/\* Player was correct, delay before playing moves \*/

delay\_ms(1000);

}

/\* player wins \*/

return 1;

}

/\* change pin mode for solenoid to an output on analog pin AO \*/

/\* add the pin mode statement here\*/

void setup()

{

pinMode(solenoid1, OUTPUT); // sets the analog pin as output

}

void loop()

{

/\* Setup IO pins and defaults \*/

ioinit();

/\* removed play\_winner(); from right here\*/

/\* Main loop \*/

while (1) {

/\* Wait for user to start game \*/

attract\_mode();

/\* Indicate the start of game play \*/

set\_leds(LED\_RED|LED\_GREEN|LED\_BLUE|LED\_YELLOW);

delay\_ms(1000);

set\_leds(0);

delay\_ms(250);

/\* Play game and handle result \*/

if (game\_mode() != 0) {

/\* Player won, play winner tones \*/

play\_winner();

}

else {

/\* Player lost, play loser tones \*/

play\_loser();

}

}

}