#include <io.h>
#include <progmem.h>
#include "global.h"
#include "uart.h"
#include "midi.h"
#include "timer.h"
#include <interrupt.h>
#include <sig-avr.h>
#include "rprintf.h"
#include "lcd.h"
#include "a2d.h"
#include "osc.h"

#define NUMPADS 8

void runMe(void) {

    //VARIABLES
    //the lastPress array and the variables starting with "o" keep track of
    //the last value sent to this MIDI channel, so we can make sure we only
    //send a new MIDI note if the value has changed. This prevents the PD
    //buffer from overflowing.
    u16 string;
    u16 ostring = 0;
    u08 pad[NUMPADS];
    u08 lastPress[NUMPADS];
    u08 squeeze, i, sfront, sback, readOff, asize, send;
    u08 osqueeze = 0;
    u08 oasize = 0;

    //INITIALIZE
    for (i=0; i<NUMPADS; i++) {
        lastPress[i] = 0;
    }

    while(1) {
        //ASIZE - this was used during development to play with some parameters
        //dynamically during development. It was not used in the final
        //instrument presentation.

        asize = a2dConvert10bit(4)>>3;  //shift me to tweak asize resolution
        if(oasize != asize) {

    }
midiNoteOnOut(asize, 0, 12);
oasize = asize;

//STRING - we convert this 10 bit number into a 7 bit one and a 3 bit
//one and then send the seven most significant bits as the midi note
//and the three least significant ones as the velocity, so if you
//divide the velocity by 2^3 and add it to the midi note, you get a
//number from 0-127 with 10 bits of resolution.

string = a2dConvert10bit(5);
send = (string != ostring);
orsstring = string;
string = string / asize;
sfront = string >> 3;
sback = (u08)string;
sback = sback << 5;
sback = sback >> 5;
if(send) {
    midiNoteOnOut(sfront, sback, 15);   //(0123456, 0000789, strChannel)
}

//SQUEEZE - the reading from the FSR in the squeeze ball
squeeze = a2dConvert10bit(2) >> 3;
if(osqueeze != squeeze) {
    midiNoteOnOut(squeeze, 0, 14);
    osqueeze = squeeze;
}

//PADS - we use two 4 to 1 multiplexors on the same chip, so first we
//determine which multiplexor (and which port) we're reading from. Odd
//pads are read from port 1 and even pads from port 0, this way we only
//have to change the input signals for every other pad. The signals are
//set as follows:
// Pad #   BitA    BitB
// 0&1     0       0
// 2&3     0       1
// 4&5     1       0
// 6&7     1       1

for(i=0; i<NUMPADS; i++) {
    if(i%2==0) {
        readOff = 0;
    } else {
        readOff = 1;
    }
}
//Set bits for Multiplexing
if(i==0) cbi(PORTB, 0);
if(i==4) sbi(PORTB, 0);
if((i%4)==0) cbi(PORTB, 1);
if((i%4)==2) sbi(PORTB, 1);
pad[i] = a2dConvert10bit(readOff) >> 3;
if (lastPress[i] != pad[i]) midiNoteOnOut(pad[i], 0, i);
lastPress[i] = pad[i];
}

int main(void) {
  uartInit();
timerInit();
a2dInit();
a2dSetReference(0x01);
midiInit();
  sei();       // enable interrupts
  outb(DDRA, 0x00);  // set pins of Port A to input (for ADC)
  outb(DDRB, 0xFF);  // set pins of Port B to output (for multiplexing)
runMe();
return 0;
}