```
//
// Modified USB and MIDI code for project from Arduino.cc
//  cisd_project_MIDIUSB_write.ino
// A simple example based on open source USB Midi software from Arduino..
// Will be incomplete, very rough semi-functional allowing students to complete and refine.
// It may even have some bugs to fix.
// This simple code is using ANSI standard C. It is used in mission-critical software. Very reliable.
// https://en.wikipedia.org/wiki/ANSI_C
//
// The windows, mac and linux IDE ( integrated development enviromnent ) is found here:
// https://www.arduino.cc/en/software
//
// The embedded microcontroller is a Atmel SAM3X8E ARM Cortex-M3 CPU is on the Arduino DUE.
// It is the is the USB 'Device'. This is a good starting point for learning ARM basics.
// https://docs.arduino.cc/hardware/due/
// https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-11057-32-bit-Cortex-M3-Microcontroller-SAM3X-S
// note: Microchip inc has merged with Atmel
//
// An open source software based subtractive synthesizer 'SURGE XT' is the USB 'Host'.
// https://surge-synthesizer.github.io/
// note: the Surge synthesizer usb cable should be unplugged prior to DUE flash programming/debugging.
// Then plug back in, then In surge re-select the DUE midi port.
// I used a usb hub so the connectors on laptop or arduino do not wear out. Other OS its unnecessary.
// The final version will not have programming cable so all of this will be unnecessary.
// It seems to be a W11 thing. Prior to updates in W11 OS this were worked properly.
// Any midi synth may be used.

#include <Arduino.h>  // default installed library
#include <MIDIUSB.h>  // add-on from arduino
#include <DueTimer.h> // add-on from arduino
// the add-on librarys are part of the IDE window on left bar. Just select 'add library' and install

// MIDI command structure used in example
// First parameter is the event type (0x09 = note on, 0x08 = note off).
// Second parameter is note-on/note-off, combined with the channel.
// Channel can be anything between 0-15. Typically reported to the user as 1-16.
// Third parameter is the note number (48 = middle C).
// Fourth parameter is the velocity (64 = normal, 127 = fastest).

int myLed = 13; // on board user debug led
int Tick =0;  // Global tick, Dont care if it overflows, only a change is used.
int LastTick =0;

//global variables 0-69  (70 minus 1) is highest DIO assignment to be used. 70 is a pad int for safety.
// unused inputs should have a pull down resistor to mininize on-chip noise.
//
// internal pullups INPUT_PULLUPS could be used during init but the progran structure would change.
// and the piezeo element cannot drive the pullups directly.
//
//

//#define DEBUG // to limit compiled code during work in progress

// an array more useable than a 'structure'.
int note_running[70]; //true false flag , is note running ?
int last_capture[70]; // keep a count of interrupts 1ms
int mton_count[70];  // min time on before turning off after triggered, a constant set during setup.

bool ledOn = false; // debug global variable

void noteOn(byte channel, byte pitch, byte velocity) {
  midiEventPacket_t noteOn = {0x09, 0x90 | channel, pitch, velocity};
  MidiUSB.sendMIDI(noteOn);
}

void noteOff(byte channel, byte pitch, byte velocity) {
  midiEventPacket_t noteOff = {0x08, 0x80 | channel, pitch, velocity};
  MidiUSB.sendMIDI(noteOff);
}
```

```
void setup() {
    delay(100); //startup safety
    int i;
  for ( i=22; i<70; i++ ) //using DIO pin numbers for clarity D22 - D69 , do not use the programmable pu
  {
     pinMode(i, INPUT); delay(1); // note: delay is a blocking function.
     } // All these now inputs, slight delay between writes as the control sect are clocked. Be safe

  // and initalize array data used
for ( i=22; i<70; i++) //
{
note_running[i] = 0; //true false , is note running ?
mton_count[i] = 25;;  // min time in milliseconds on before turning off, this in a input debounce scheme
last_capture[i] = 0;;  // min time on before turning off, this in a input debounce scheme
}


// set up timer
    pinMode(myLed, OUTPUT);
    Timer3.attachInterrupt(myHandler);
    Timer3.start(1000); // interrupts every 1 millisecond ( 1,000 microseconds )

// set up debug port
  Serial.begin(115200);
    delay(10); //  safe delay for uart clock

}

//interrupt timer 1ms

void myHandler(){
  Tick++; // Global variable maybe delete later
    ledOn = !ledOn; // debug led 500 hz
    digitalWrite(myLed, ledOn); // Led on, off, on, off...
}


// First parameter is the event type (0x0B = control change).
// Second parameter is the event type, combined with the channel.
// Third parameter is the control number number (0-119).
// Fourth parameter is the control value (0-127).

void controlChange(byte channel, byte control, byte value) {
  midiEventPacket_t event = {0x0B, 0xB0 | channel, control, value};
  MidiUSB.sendMIDI(event);
}

void loop() {
  int Statetmp; // local variable in loop
  int i;
  #ifdef DEBUG // single channel input crude functional test uses blocking delay and printf debug writes
while(1){  // inner loop never exits, if it does the software is broken
  Statetmp = digitalRead(53);// D53 used in demo mov
  if(Statetmp == 1) //using true logic, 1 is a logic high, external 1Meg pulldown resistor.
  {

  //Serial.println("Sending note on");
  noteOn(0, 48, 64);   // Channel 0, middle C, normal velocity
  MidiUSB.flush();//this may not be necessary. have to look at flush code.
  delay(25);
  //Serial.println("Sending note off");
  noteOff(0, 48, 64);  // Channel 0, middle C, normal velocity
  MidiUSB.flush(); //this may not be necessary. have to look at flush code.
    delay(10); // testin what actual time outs need to be
  }
} //inner forever loop
#endif

#ifndef DEBUG // normal flow not debug
while(1){  // never exits, if it does the software is broken
```

```
  for ( i=22; i<70; i++ ) //using DIO pin numbers for clarity D22 - D69
  {
    Statetmp = digitalRead(i);
    // Statetmp = digitalRead(53);// D53 testing same as used in demo mov
    if(  (Statetmp == 1) && (note_running[i]==0)  ){ //note went on, send to midi
      note_running[i]= 1; //
      last_capture[i] = 0; //reset
      noteOn(0, 48, 64);   // Channel 0, middle C, normal velocity
    }

    if ((Statetmp == 0 ) && (last_capture[i] >= 25) ){ // time to send off to synth
      last_capture[i] = 0; // reset
      note_running[i]= 0; // reset too
     noteOff(0, 48, 64);  // Channel 0, middle C, normal velocit
    }

    if(Tick != LastTick) { // 1ms timer capture update
      LastTick = Tick;
      last_capture[i]++;    } // TO DO add a variable/test for idle state so no increment if idle.
  }
} //inner forever loop
 #endif

  // controlChange(0, 10, 65); // Un-used un-tested ,Set the value of controller 10 on channel 0 to 65
  }// main
```