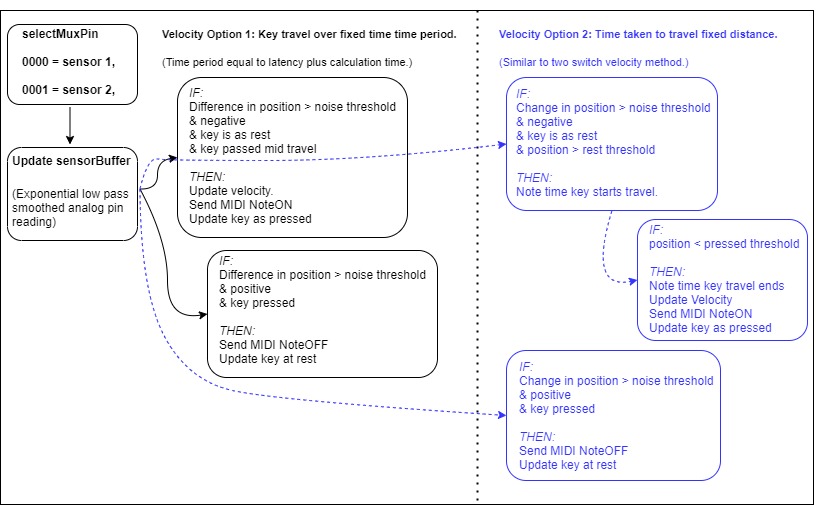
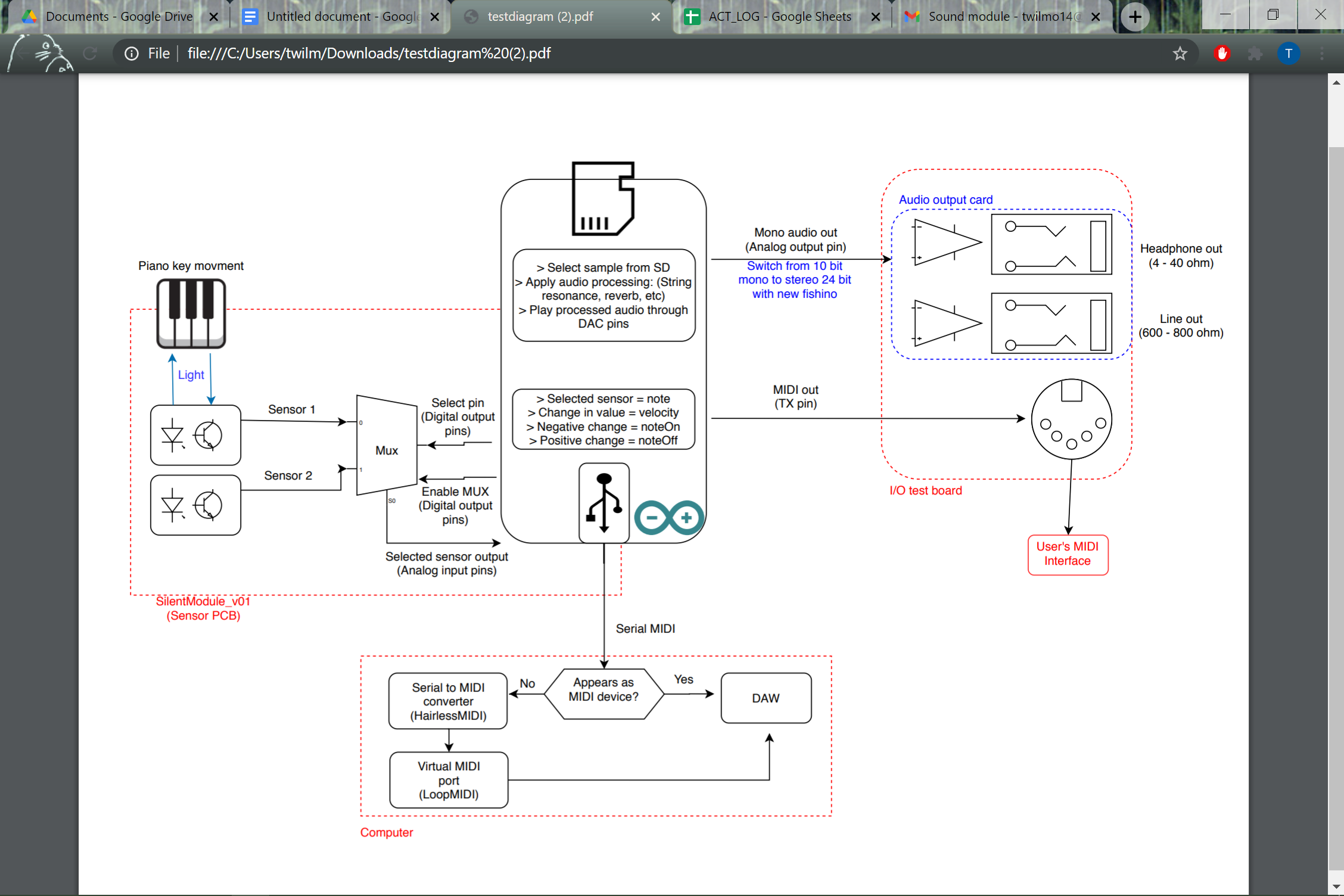
MIDI out code flow, 28th December 2020



Sound Module Concept Flowchart, 19th November 2020  


List of TW daily notes/tasks from ACT\_LOG, October-December 2020

|  |  |
| --- | --- |
| 01/10/20 | Designed silent module initial prototype 1 for sensor CNY70 with Arduino uno.  Designed initial silent module prototype 2 for MUX 16:1. |
| 02/10/20 | Built silent module initial prototype 1 |
| 05/10/20 | Tested sensor placement with outcome of +/- 40 (10bit adc) velocity values at non optimal placement. Noise level +/- 3 |
| 06/10/20 | Tested sensor limitations.  Circuit redesign with desired components. Established that key colour and ambient light will not affect sensor.  Wrote code to find precise optimal key placement; outcome of +/-100. velocity values. Noise +/- 2. |
| 07/10/20 | Coding single sensor to midi.  Rewrite code for 7mm lever travel.  Test circuit compatibility for 3.3V 12bit ADC boards.  Outline plan for October. |
| 08/10/20 | Testing sensor height guide code with Keybird piano.  For 10 bit ADC key at rest = 960 (on 0-1023 scale) key pressed = 710~. Outcome +/- 125~ velocity values.  Began scaling sensor output to velocity values (0-127).  Emitter resistor switch from 220Ω to 420Ω to account for 7mm lever travel. |
| 09/10/20 | Tested polyphony with 2 sensors.  Variations in emitter resistor values of 20Ω or a height difference of 1mm is enough to result in inaccurate velocity ratings using a 10 bit ADC.  Booked an MKR 12 bit ADC board for collection. Testing adg608br MUX 8:1 chips and rewriting code. |
| 12/10/20 | Picked up 12 bit adc arduino MKR 1010 board and other components.  Circuit design and coding for cascading MUX 8;1 and 16:1 (method for connecting multiple sensors using bit code addresses per key 0-69). |
| 13/10/20 | Emitter resistor switched to 2.7kΩ after iterative testing. This focuses the sensor to 1 key only and allows it to be placed as close as possible to lever without contact.  Outcome is a sensor range between 4050 to 1020 (12bit ADC scale 0-4095 on MKR 1010) with velocity value range of +/-1500~2000 to be scaled to 0-127 midi range.  This range/resolution allows noise and variance between sensor circuits to be negated and gives a more accurate representation of velocity.  Rewrote sensor placement guide code for 12 bit ADC scale. |
| 14/10/20 | Built silent module initial prototype 2. Reading from 8 sensors via ADG608BR. Debugging crosstalk between channels. |
| 15/10/20 | Configuring circuit design and software encoding multiple MUX 16:1 / 8:1 for 69 keys.  Optimised circuit design to activate/read one of five MUX 16:1 in series via En pins. |
| 16/10/20 | 8 key working build with adg608 without latency, full polyphony. Iterative hardware testing and debugging. |
| 19/10/20 | Picked up leads and resistors.  Testing circuit and debugging hardware (Adg608 crosstalk issues due to loose connection) |
| 20/10/20 | Finalised fine tuning of resistors and circuit design. Early PCB sketches. |
| 21/10/20 | Continued work on PCB design |
| 22/10/20 | Testing series connected/encoded mux 8:1 boards and PCB redesigns for key placement.  Testing 69 key (in groups of 8) encoding of hardware using active HIGH encoder/activation pins of ADG608 in series with sensors under keys (simplifiying PCB and minimising board size/cost). |
| 23/10/20 | PCB redesigns with updated sensor placement and minimised footprint. Designing for 16:1 MUX board due to available pins on MKR 1010.  Checked power options. LiPo 2000 mAh rechargable battery pack available for MKR 1010 board and can be connected without PCB (ie replaceble after it degrades over years). |
| 25/10/20 | Moved to final board layout. Checking for errors. Moving components to final positions then running top and bottom copper paths.  Noticed critical error during translation caused by CNY70 escheema schematic symbol mismatch with real component (Emitter and collector pins were reversed i.e symbol designer counted clockwise 1-2-4-3).  Rectification requires redesign prior to MUX 16:1 boards, including positioning of 69 sensors + resistors again. |
| 26/10/20 | Created part symbol/library for CNY70 as CNY70\_TW, correcting pin issues. Updated/replaced all sensors in design.  Restarted PCB component layout and copper paths. Double-checked tracks and pads match Elecrow requirements (printing company).  Redesigning board for split from 920mm length into 2 smaller parts due to maximum printing length of 550mm. Current plan is to connect two boards with through holes and male to male leads.  Majority of time spent ensuring sensors are accurately positioned after each redesign. |
| 27/10/20 | Checking options for later additions/user I/O without PCB via arduino including pedal controls, midi out, battery packs, control buttons.  Finalised split board design, parts layout, labelling and PCB (front and back copper tracks, screw holes etc).  Kicad defaults should be compatible with Elecrow but final checks are recommended before tomorrow's order. |
| 28/10/20 | Finalised design checks, matched Elecrow standards and produced Elecrow compatible gerbers (PCB build instructions).  Got quotes from Elecrow for pcb. |
| 29/10/20 | Ordering PCB.  Working on final software and assembly guides in preparation of parts arrival. |
| 30/10/20 | Solving issues with PCB order.  Work on placement/assembly software  Meeting Viktor to discuss future of Silent module project |
| 31/10/20 | Collecting PCB pdf, schematics and images from October 2020 of project from initial prototype 1 and 4. |
| 02/11/20 | Ordering RS components via Ben for 1 full PCB build.  Investigating options for quick-to-make/outsource production model of silent module using primarily surface mounted components.  Uploading kicad project for Viktor to download. |
| 03/11/20 | Preparing demo prototype for Victor and Marcus  Planning tasks/exercises to do with Victor  Redoing PCB sensor symbol library after problem with downloading v01\_rTW04.  Debugging Arduino board not recognised over USB |
| 04/11/20 | Solved issue with Arduino sketch upload over usb. (Press reset button twice to enable bootloader then upload. Repeat)  Demonstrating early prototype and circuit design to Victor  Working on debugging Kicad (Symbol libraries would not appear automatically)  Discussing division of tasks and project future. |
| 06/11/20 | Discussing sound module I/O and sampling vs synth.  Getting tools from Marcus.  Setting up PCB and parts.  Reconfiguring software to account for board split and optimising for low latency. |
| 10/11/20 | Moved piano back to hall of the odd.  Set up soldering station.  Checked method with Victor.  Shopped for fume extractor.  Finalised Placement guide/sensor test and Velocity test (rTW12) software.  Set up shared github project and repository with Victor. |
| 12/11/20 | Moved project into "organisation" on github (fixing issue with hidden shared cards).  Plan for I/O requirments and project outline.  Coding velocity and note values over MIDI for USB and MIDI connector.  Investigating ways to eliminate list of software required to turn Arduino usb serial data into MIDI device |
| 17/11/20 | Troubleshooting MIDI software.  Chasing components and checking I/O is ready for prototyping. |
| 18/11/20 | Serial to MIDI converter (HairlessMIDI) then MIDI port created by LoopMIDI working robustly after checks.  Making USB device appear as MIDI device should work via MIDIUSB.h library, however returns "ftdi driver" error.  (Research suggests hacking drivers to allow 31250 baud rate but this is not a viable solution).  Collected parts from DTU Skylab and showed Victor where to find parts/packages. |
| 19/11/20 | Created flow chart for explaining current project and how all parts shoud work.  Filling github projects with detailed task cards (SoundModulev01, I/O board, KeybirdSilentModulev01). |
| 23/11/20 | Collecting MUX and buying parts from elextra |
| 25/11/20 | Silent Module PCB tests |
| 26/11/20 | Temporary mounted silent module treble and troubleshooting hardware.  Soldering/troubleshooting PCB hardware issues (MUX requires 5V, unlike earlier prototypes).  Reading 16 sensors.  Confirmed noise level, sensor values at rest and pressed match prototypes. |
| 27/11/20 | Reading all 69 sensors without latency.  (all enable pins on mux must be set to low)  Confirmed switching MUX noise is not an issue.  Solved issue with double hits caused by MIDI velocity updating too quickly. |
| 28/11/20 |  |
| 29/11/20 |  |
| 30/11/20 | Solved latency issues (1.computational delay for velocity calculations plus 2. buffering using "delay()" *(freezes code)* instead of "millis()" *(gives time in ms)* resulted in clock mismatch).  Changed velocity algorithm from "distance key travelled during fixed time period" to "time taken to cover fixed distance"  (removing issues with latency and giving more accurate velocity) |
| 01/12/20 | Producing sound via MIDI outconnection without latency  Producing midi messages over USB without latency  Troubleshooting code and pcb whilst combining Treble and Bass PCBs |
| 02/12/20 | Mounting block and PCB fitting requirements/redesigns  Introduced logic level converter to enable MIDI out to sound module and over USB from MKR board |
| 03/12/20 |  |
| 04/12/20 | Testing velocity with bass board.  No latency with either midi out or usb. |
| 05/12/20 |  |
| 06/12/20 |  |
| 07/12/20 | Tested software solutions to noise handling on board.  Result: Sensor noise is caused by poor grounding and lack of power supply filtering on hardware.  Software solutions: A moving average filter with window upto 6 samples before introducing latency. |
| 08/12/20 | Applied moving average filter.  Debugged power supply vs soldering issues |
| 09/12/20 | Divided treble and bass pcb.  Resoldered PCB.  Isolated main source of noise as interference across MUX lines |
| 10/12/20 |  |
| 11/12/20 | Pcb soldering and fault finding (sensors on mux pin 23 affected by Vcc mod). (Tolerance on 2.7k is acceptable but 110k varies enough to have noticeable effect).  To work around ground interference, bridge unconnected mux COM to ground. (for v02 low pass RC circuits should be used alongside diodes)  After callibration and code updates, the MIDI note off / release now works as intended.  Levelled and calibrated silent module with piano using software placement guide and measuring callipers.  (Method:  1. Raise silent module until noticeable clicking noise from lever contact on key release.  2, Reduce height until point when clicking contact noise is eliminated.  3. Use calliper to find mid key travel or press hammer against string.  4. Adjust the silent module height so that the sensor readout difference between lowest and highest keys is minimal.  5. Update "pressedPositionThresh" as the maximum sensor value when key is at mid travel point. (Current aim 2500 on 12 bit adc) |
| 14/12/20 | Testing velocity algorithm |
| 15/12/20 | Balancing sensitivity against speed of play |
| 16/12/20 | Meeting with DSP candidate |

Sensor placement guide code

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Title: Sensor readout test and placement guide for Keybird Silent Module v01  Author: Thomas Wilmot  Date modified: 08/12/20  Version: v01  Revision: rTW15  Notes: Designed for Arduino MKR 1010 with IDE version 1.8.13.  Values are set based on 12 Bit ADC resolution.  Calibration method:  1. Raise silent module until noticeable clicking noise from lever contact on key release.  2, Reduce height until point when clicking contact noise is eliminated.  3. Use calliper to find mid key travel or press hammer against string.  4. Adjust the silent module height so that the sensor readout difference between lowest and highest keys is minimal.  5. Update "pressedPositionThresh" as the maximum sensor value when key is at mid travel point. (Current aim 2500 on 12 bit adc)  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  // Pin Definitions  const int selectPins[4] = {0, 1, 2, 3}; // S0~0, S1~1, S2~2 s3-3  // Exponential filter (smooths out sensor noise)  /\*\*\*\*\*\* Updates key position with weight towards previous values in buffer\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\*\*\*\* Reduce 'alphaW' to increase smoothing but also reduces sensitivity to fast playing\*\*/  const int numReadings = 3 ; // Buffer size  const float alphaW = 0.45; // Smoothing ratio (0 = High, 0.5 = Low)  int sensorBuffer[numReadings] ; // Buffer for sensor values  int totalBuffer[69] ; // Sum of values in buffer  int bufferIndex ; // Index of where in the buffer a sensor reading is  int bufferAverage[69] ; // Average of buffered values in for key 0 - 68  // Clock Values (in microseconds)  unsigned long timeKeyStartTravel[69] ; // Time key press starts  unsigned long timeKeyEndTravel[69] ; // Time key fully pressed  unsigned long timeForKeyTravel[69] ; // Time to complete key travel  const int timeMidRead = 100 ; // Time between buffer readings  // Velocity Settings  /\*\*\*\*\*\* To increase velocity sensitivity, decrease slowestKeyTravel and/or increase fastestKeyTravel. \*\*\*\*\*\*\*/  /\*\*\*\*\*\* Increase noise threshold if false hits or keys to the sides are being detected \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  int positionVal1[69] ; // Buffered first position values  int difVal[69] ; // Buffered differnce in position  int velocityVal[69] ; // Buffered velocity values  int keyStatus[69] ; // Register for key pressed or at rest (1 = pressed, 0 = at rest)  int veloSensitivity = 2400 ; // Velocity sensitivity (maximum expected change in key position) (2000 = low sensitivity, 1500~ = medium)  int noiseThresh = 50 ; // Noise threshold for change in key position (100~ = expected, 0 = Ground)  int slowestKeyTravel = 80000; // Slowest time from key rest to fully pressed (in microseconds) = 0 Velocity  int fastestKeyTravel = 200; // Fastest time from key rest to fully pressed (in microseconds) = 127 Velocity  int restPositionThresh = 3600 ; // Threshold above which key is considered at rest (note off)  int pressedPositionThresh = 2200 ; // Threshold below which key is considered pressed (note on)  void setup()  {  Serial.begin(31250); // Initialize the serial port (baud rate determined by MIDI standards)  analogReadResolution(12); // Set ADC resolution to 12 bit    // Set up the select pins as outputs:  for (int i = 0; i < 4; i++)  {  pinMode(selectPins[i], OUTPUT);  digitalWrite(selectPins[i], HIGH);  }  }  // The selectMuxPin function sets the S0, S1, s2 and S3 pins  // accordingly, given a sensor pin from 0-15.  // e.g. 0000 = sensor 1, 0001 = sensor 2, 0010 = sensor 3... etc  void selectMuxPin(byte pin)  {  for (int i = 0; i < 4; i++)  {  if (pin & (1 << i))  digitalWrite(selectPins[i], HIGH);  else  digitalWrite(selectPins[i], LOW);  }  }  void loop()  {    // Sensor Placement Guide Levels (12bit)  Serial.print(4095); // Clipping signal (sensor too close) +5V  Serial.print("\t");  Serial.print(3600); // Good level at key rest  Serial.print("\t");  Serial.print(2200); // Mid press level  Serial.print("\t");  Serial.print(1000); // Key pressed (+/-100)  Serial.print("\t");  Serial.print(0); // Ground 0V  Serial.print("\t");  /\*  // Sensor Placement Guide Levels (10bit) \*\*\*\*\*\*\*\*needs checking\*\*\*\*\*\*\*\*\*  Serial.print(1095); // Clipping signal (sensor too close)  Serial.print("\t");  Serial.print(1000); // Good level at key rest  Serial.print("\t");  Serial.print(100); // Key pressed  Serial.print("\t");  Serial.print(0); // Ground or no signal  Serial.print("\t");  \*/  // Enable all MUX  // digitalWrite(7 , LOW); // Activate MUX by set En pin LOW (0) = Ground  // digitalWrite(8 , LOW); // Deactivate all other MUX by setting En pins HIGH (1) = +3.3V  // digitalWrite(9 , LOW);  // digitalWrite(10 , LOW);  // digitalWrite(11 , LOW);    // Loop through all sensors.  // Key  for (byte pin = 10; pin <= 15; pin++) // Loop through sensor pins on MUX  {  selectMuxPin(pin); // Select one sensor at a time  int keyNum = int(pin) + 21 ; // Key number (counting from 0 to 68)  // int inputValue = analogRead(A3); // Read MUX A  // Serial.print(String(pin-6) + "a" + "\t" + String(inputValue) + "\t"); // Display Position Value (Key number+Mux letter\_tab\_Position value\_tab)    //totalBuffer[keyNum] = bufferAverage[keyNum] \* 0.9 ; // Reset sum of buffer values  for (bufferIndex = 0; bufferIndex < numReadings; bufferIndex++)  {    sensorBuffer[bufferIndex] = analogRead(A3); // read from the sensor:  totalBuffer[keyNum] = (alphaW\*sensorBuffer[bufferIndex]) + ((1-alphaW)\*(totalBuffer[keyNum])) ; // add the reading to the total:  }  if (bufferIndex >= numReadings) // if we're at the end of the buffer...  {  bufferIndex = 0; // ...wrap around to the beginning:  bufferAverage[keyNum] = totalBuffer[keyNum] ;// numReadings; // calculate and update the average  }  Serial.print(String(pin+21) + "a" + "\t" + String(bufferAverage[keyNum]) + "\t"); // Display Position Value (Key number+Mux letter\_tab\_Position value\_tab)  }    // Key 38 to 50  for (byte pin = 0; pin <= 12; pin++)  {  selectMuxPin(pin);  int keyNum = int(pin) + 37 ; // Key number (counting from 0 to 68)  //int inputValue = analogRead(A2); // read MUX B  //Serial.print(String(pin+10) + "b" + "\t" + String(inputValue) + "\t");    for (bufferIndex = 0; bufferIndex < numReadings; bufferIndex++)  {  sensorBuffer[bufferIndex] = analogRead(A4); // read from the sensor:  totalBuffer[keyNum] = (alphaW\*sensorBuffer[bufferIndex]) + ((1-alphaW)\*(totalBuffer[keyNum])) ; // add the reading to the total:  }  if (bufferIndex >= numReadings) // if we're at the end of the buffer...  {  bufferIndex = 0; // ...wrap around to the beginning:  bufferAverage[keyNum] = totalBuffer[keyNum] ;// /numReadings; // calculate and update the average  }  Serial.print(String(pin+37) + "b" + "\t" + String(bufferAverage[keyNum]) + "\t");  }    /\*  // Key 26 to 37  for (byte pin = 4; pin <= 15; pin++)  {  selectMuxPin(pin);  int inputValue = analogRead(A3); // read MUX C  Serial.print(String(pin+22) + "c" + "\t" + String(inputValue) + "\t");  }    // Key 38 to 53  for (byte pin = 0; pin <= 15; pin++)  {  selectMuxPin(pin);  int inputValue = analogRead(A4); // read MUX D  Serial.print(String(pin+38) + "d" + "\t" + String(inputValue) + "\t");  }  // Key 54 to 69  for (byte pin = 0; pin <= 15; pin++)  {  selectMuxPin(pin);  int inputValue = analogRead(A5); // read MUX E  Serial.print(String(pin+54) + "e" + "\t" + String(inputValue) + "\t");  }  \*/  Serial.println(); // New line    } |

MIDI out code with two velocity algorithm options

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Title: MIDI out test for Keybird Silent Module v01  Author: Thomas Wilmot  Date modified: 15/12/20  Version: v01  Revision: rTW17  Notes: Designed for Arduino MKR 1010 with IDE version 1.8.13.  Values are set based on 12 Bit ADC resolution.  To send MIDI, attach a MIDI out jack (female DIN-5) to Arduino.  DIN-5 pinout is: \_\_\_\_\_\_\_  pin 2 - Gnd / \  pin 4 - 220 ohm resistor to +5V | 3 1 | MIDI jack  pin 5 - Arduino /14 (TX) | 5 4 |  all other pins - unconnected \\_\_\_2\_\_\_/  Note/pitch starting on E, Keybird 1-69 = Midi 28-96  Create com to MIDI port via LoopMIDI  Convert serial data to MIDI via HairlessMIDI  Play notes via generic virtual keyboard  // MIDI functions  // First parameter is the event type (0x90 = note on, 0x80 = note off).  // Second parameter is note number ( e.g. 48 = middle C).  // Third parameter is the velocity (64 = normal, 127 = fastest).  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  // Arduino Pin Definitions (for MUX)  const int selectPins[4] = {0, 1, 2, 3}; // S0~0, S1~1, S2~2 s3-3  // Clock Values (in microseconds)  unsigned long timeKeyStartTravel[69] ; // Time key press starts  unsigned long timeKeyEndTravel[69] ; // Time key fully pressed  unsigned long timeForKeyTravel[69] ; // Time to complete key travel  // Velocity Settings  /\*\*\*\*\*\* To increase velocity sensitivity, decrease slowestKeyTravel and/or increase fastestKeyTravel. \*\*\*\*\*\*\*/  /\*\*\*\*\*\* Increase noise threshold if false hits or keys to the sides are being detected \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  int positionVal1[69] ; // Buffered first position values  int difVal[69] ; // Buffered differnce in position  int velocityVal[69] ; // Buffered velocity values  int keyStatus[69] ; // Register for key pressed or at rest (1 = pressed, 0 = at rest)  int veloSensitivity = 2750 ; // Velocity sensitivity (maximum expected change in key position) (2000 = low sensitivity, 1500~ = medium)  int noteThresh = 800 ; // Minimum change in position before note is registered (600~ = expected)  int noiseThresh = 40 ; // Noise threshold for change in key position (100~ = expected, 0 = Ground)  int slowestKeyTravel= 80000; // Slowest time from key rest to fully pressed (in microseconds) = 0 Velocity  int fastestKeyTravel = 200; // Fastest time from key rest to fully pressed (in microseconds) = 127 Velocity  int restPositionThresh = 3600 ; // Threshold above which key is considered at rest (note off)  int pressedPositionThresh = 2200 ; // Threshold below which key is considered pressed (note on)  // Smoothing filters  /\*\*\*\*\*\* Updates key position with weight towards previous values in buffer\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\*\*\*\* Reduce 'alphaW' to increase smoothing but also reduces sensitivity to fast playing\*\*/  const int numReadings = 5 ; // Buffer size  const float alphaW = 0.4; // Smoothing sensor ratio (0 = High, 0.5 = Low)  const float alphaW2 = 0.9; // Smoothing difference ratio (0.5 = High, 1 = Low)  int sensorBuffer[numReadings] ; // Buffer for sensor values  int totalBuffer[69] ; // Sum of values in buffer  int bufferIndex ; // Index of where in the buffer a sensor reading is stored  int bufferAverage[69] ; // Average of buffered values in for key 0 - 68  void setup()  {  Serial.begin(31250); // Initialize the serial port on USB (baud rate determined by MIDI standards)  Serial1.begin(31250); // Initialize the serial port ON TX pin / MIDI out connection  analogReadResolution(12); // Set ADC resolution to 12 bit    // Set up the select pins as outputs:  for (int i = 0; i < 4; i++)  {  pinMode(selectPins[i], OUTPUT);  digitalWrite(selectPins[i], HIGH);  }  }  // The selectMuxPin function sets the S0, S1, s2 and S3 pins  // accordingly, given a pin from 0-15.  // e.g. 0000 = sensor 1, 0001 = sensor 2, 0010 = sensor 3  void selectMuxPin(byte pin)  {  for (int i = 0; i < 4; i++)  {  if (pin & (1 << i))  digitalWrite(selectPins[i], HIGH);  else  digitalWrite(selectPins[i], LOW);  }  }  void loop()  {      // Enable all MUX    digitalWrite(7 , LOW);  digitalWrite(8 , LOW);  digitalWrite(9 , LOW);  digitalWrite(10 , LOW);  digitalWrite(11 , LOW);    // Loop through all sensors.  // Key treble mux c  for (byte pin = 10; pin <= 15; pin++) // Loop through sensor pins on MUX  {  selectMuxPin(pin); // Select one sensor at a time  int keyNum = int(pin) +21 ; // Key number (counting from 0 to 68)  int note = keyNum + 28 ; // MIDI Key number (e.g. middle C = 48)  //int inputValue = analogRead(A3); // Read MUX C  for (bufferIndex = 0; bufferIndex < numReadings; bufferIndex++) // Sensor position buffer  {  sensorBuffer[bufferIndex] = analogRead(A3); // read from the sensor  totalBuffer[keyNum] = (alphaW\*sensorBuffer[bufferIndex]) + ((1-alphaW)\*(totalBuffer[keyNum])) ; // add the weighted reading to the total  }  if (bufferIndex >= numReadings) // if we're at the end of the buffer...  {  bufferIndex = 0; // ...wrap around to the beginning:  bufferAverage[keyNum] = totalBuffer[keyNum] ; // and update the average  }  difVal[keyNum] = (alphaW2\*(bufferAverage[keyNum] - positionVal1[keyNum])) - ((1-alphaW2)\*(difVal[keyNum])); // Change in key position  // Serial.print(String(pin+21) + "b" + "\t" + String(difVal[keyNum]) + "\t");  if (difVal[keyNum] < (-1\*noiseThresh) && keyStatus[keyNum] == 0) // Note On (If negative change past noise threshold and note is off)  {  if( bufferAverage[keyNum] > restPositionThresh ) // If key position near resting threshold...  {  timeKeyStartTravel[keyNum] = micros() ; // Then store time key travel has begun  }  if( bufferAverage[keyNum] < pressedPositionThresh && difVal[keyNum] < (-0.8\*noteThresh)) // If key position past fully pressed threshold  {  timeKeyEndTravel[keyNum] = micros() ; // Store time key crosses pressed threshold  timeForKeyTravel[keyNum]=(timeKeyEndTravel[keyNum]-timeKeyStartTravel[keyNum]); // Time taken for key to travel from rest to fully pressed  keyStatus[keyNum] = 1 ; // Register that key is pressed (prevents double hits)  velocityVal[keyNum] =map(difVal[keyNum], (-1\*noteThresh), (-1\*veloSensitivity), 0, 127); // Velocity = Mapped change in position during calculation time (can lag)  //map(timeForKeyTravel[keyNum], slowestKeyTravel, fastestKeyTravel, 0, 127) ; // Velocity = Mapped time taken to complete key travel (fixed distance)  //  noteOn(0x90, note, velocityVal[keyNum]); // MIDI message see 'void noteOn' below  }  }  if (difVal[keyNum] > (noteThresh/3) && keyStatus[keyNum] == 1) // Note Off (If positive change past noise threshold and note is note is on)  {  keyStatus[keyNum] = 0 ; // Register note is off  velocityVal[keyNum] = 0 ; // Velocity = 0  noteOff(0x80, note, velocityVal[keyNum]); // MIDI message see 'void noteOff' below  }  positionVal1[keyNum] = bufferAverage[keyNum] ; // Update reference position value after calculations  }    // Key treble mux d  for (byte pin = 0; pin <= 12; pin++)  {  selectMuxPin(pin);  int keyNum = int(pin) + 37 ;  int note = keyNum + 28 ;  // int inputValue = analogRead(A4); // read MUX B  for (bufferIndex = 0; bufferIndex < numReadings; bufferIndex++) // Sensor position buffer  {  sensorBuffer[bufferIndex] = analogRead(A4); // read from the sensor  totalBuffer[keyNum] = (alphaW\*sensorBuffer[bufferIndex]) + ((1-alphaW)\*(totalBuffer[keyNum])) ; // add the weighted reading to the total  }  if (bufferIndex >= numReadings) // if we're at the end of the buffer...  {  bufferIndex = 0; // ...wrap around to the beginning:  bufferAverage[keyNum] = totalBuffer[keyNum] ; // and update the average  }  difVal[keyNum] = (alphaW2\*(bufferAverage[keyNum] - positionVal1[keyNum])) - ((1-alphaW2)\*(difVal[keyNum])); // Change in key position  //Serial.print(String(pin+37) + "b" + "\t" + String(difVal[keyNum]) + "\t");    if (difVal[keyNum] < (-1\*noiseThresh) && keyStatus[keyNum] == 0) // Note On (If negative change past noise threshold and note is off)  {  if( bufferAverage[keyNum] > restPositionThresh ) // If key position near resting threshold...  {  timeKeyStartTravel[keyNum] = micros() ; // Then store time key travel has begun  }  if( bufferAverage[keyNum] < pressedPositionThresh && difVal[keyNum] < (-0.8\*noteThresh)) // If key position past fully pressed threshold  {  timeKeyEndTravel[keyNum] = micros() ; // Store time key crosses pressed threshold  timeForKeyTravel[keyNum]=(timeKeyEndTravel[keyNum]-timeKeyStartTravel[keyNum]); // Time taken for key to travel from rest to fully pressed  keyStatus[keyNum] = 1 ; // Register that key is pressed (prevents double hits)  velocityVal[keyNum] =map(difVal[keyNum], (-1\*noteThresh), (-1\*veloSensitivity), 0, 127); // Velocity = Mapped change in position during calculation time (can lag)  //map(timeForKeyTravel[keyNum], slowestKeyTravel, fastestKeyTravel, 0, 127) ; // Velocity = Mapped time taken to complete key travel (fixed distance)  //  noteOn(0x90, note, velocityVal[keyNum]); // MIDI message see 'void noteOn' below  }  }  if (difVal[keyNum] > (noteThresh/3) && keyStatus[keyNum] == 1) // Note Off (If positive change past noise threshold and note is note is on)  {  keyStatus[keyNum] = 0 ; // Register note is off  velocityVal[keyNum] = 0 ; // Velocity = 0  noteOff(0x80, note, velocityVal[keyNum]); // MIDI message see 'void noteOff' below  }  positionVal1[keyNum] = bufferAverage[keyNum] ; // Update reference position value after calculations  }  //Serial.println();  /\*  // Key 26 to 37  for (byte pin = 4; pin <= 15; pin++)  {  selectMuxPin(pin);  int keyNum = int(pin) + 21 ;  int note = keyNum + 28 ;  int inputValue = analogRead(A3); // read MUX C  difVal[keyNum] = (inputValue - positionVal1[keyNum]);  if (difVal[keyNum] < (-1\*noiseThresh) && keyStatus[keyNum] == 0)  {  if( inputValue > restPositionThresh )  {  timeKeyStartTravel[keyNum] = micros() ;  }  if( inputValue < pressedPositionThresh )  {  timeKeyEndTravel[keyNum] = micros() ;  timeForKeyTravel[keyNum] = (timeKeyEndTravel[keyNum] - timeKeyStartTravel[keyNum] );  keyStatus[keyNum] = 1 ;  velocityVal[keyNum] = map(difVal[keyNum], (-1\*noiseThresh), (-1\*veloSensitivity), 0, 127);  //map(timeForKeyTravel[keyNum], slowestKeyTravel, fastestKeyTravel, 0, 127) ;  //  noteOn(0x90, note, velocityVal[keyNum]);  }  }  if (difVal[keyNum] > noiseThresh && keyStatus[keyNum] == 1)  {  keyStatus[keyNum] = 0 ;  timeForKeyTravel[keyNum] = 0 ;  velocityVal[keyNum] = 0 ;  noteOff(0x80, note, velocityVal[keyNum]);  }  positionVal1[keyNum] = {inputValue} ;  }    // Key 38 to 53  for (byte pin = 0; pin <= 15; pin++)  {  selectMuxPin(pin);  int keyNum = int(pin) + 37 ;  int note = keyNum + 28 ;  int inputValue = analogRead(A4); // read MUX D  difVal[keyNum] = (inputValue - positionVal1[keyNum]);  if (difVal[keyNum] < (-1\*noiseThresh) && keyStatus[keyNum] == 0)  {  if( inputValue > restPositionThresh )  {  timeKeyStartTravel[keyNum] = micros() ;  }  if( inputValue < pressedPositionThresh )  {  timeKeyEndTravel[keyNum] = micros() ;  timeForKeyTravel[keyNum] = (timeKeyEndTravel[keyNum] - timeKeyStartTravel[keyNum] );  keyStatus[keyNum] = 1 ;  velocityVal[keyNum] = map(difVal[keyNum], (-1\*noiseThresh), (-1\*veloSensitivity), 0, 127);  //map(timeForKeyTravel[keyNum], slowestKeyTravel, fastestKeyTravel, 0, 127) ;  noteOn(0x90, note, velocityVal[keyNum]);  }  }  if (difVal[keyNum] > noiseThresh && keyStatus[keyNum] == 1)  {  keyStatus[keyNum] = 0 ;  timeForKeyTravel[keyNum] = 0 ;  velocityVal[keyNum] = 0 ;  noteOff(0x80, note, velocityVal[keyNum]);  }  positionVal1[keyNum] = {inputValue} ;    }  // Key 54 to 69  for (byte pin = 0; pin <= 15; pin++)  {  selectMuxPin(pin);  int keyNum = int(pin) + 53 ;  int note = keyNum + 28 ;  int inputValue = analogRead(A5); // read MUX E  difVal[keyNum] = (inputValue - positionVal1[keyNum]);  if (difVal[keyNum] < (-1\*noiseThresh) && keyStatus[keyNum] == 0)  {  if( inputValue > restPositionThresh )  {  timeKeyStartTravel[keyNum] = micros() ;  }  if( inputValue < pressedPositionThresh )  {  timeKeyEndTravel[keyNum] = micros() ;  timeForKeyTravel[keyNum] = (timeKeyEndTravel[keyNum] - timeKeyStartTravel[keyNum] );  keyStatus[keyNum] = 1 ;  velocityVal[keyNum] = map(timeForKeyTravel[keyNum], slowestKeyTravel, fastestKeyTravel, 0, 127) ;  //map(difVal[keyNum], (-1\*noiseThresh), (-1\*veloSensitivity), 0, 127);  noteOn(0x90, note, velocityVal[keyNum]);  }  }  if (difVal[keyNum] > noiseThresh && keyStatus[keyNum] == 1)  {  keyStatus[keyNum] = 0 ;  timeForKeyTravel[keyNum] = 0 ;  velocityVal[keyNum] = 0 ;  noteOff(0x80, note, velocityVal[keyNum]);  }  positionVal1[keyNum] = {inputValue} ;  }  \*/  delay(1);  }    void noteOn(int cmd, int pitch, int velocity) {  Serial.write(cmd); // MIDI command e.g. note on = 0x90 and note off = 0x80  Serial.write(pitch); // MIDI note e.g. middle C = 48  Serial.write(velocity); // MIDI velocity 127 = fastest key press, 0 = slowest key press or not press  Serial1.write(cmd); // Same as above for TX (Midi out connection)  Serial1.write(pitch);  Serial1.write(velocity);  }  void noteOff(int cmd, int pitch, int velocity) {  Serial.write(cmd);  Serial.write(pitch);  Serial.write(velocity);    Serial1.write(cmd); // Same as above for TX (Midi out connection)  Serial1.write(pitch);  Serial1.write(velocity);  } |