```
// Scan Makin Great Manual and Pistons with Arduino Leonardo board
// Using MIDIUSB library v.1.0.4 by Gary Grewal
// John Harvey May 2019

// Loop period 7.2ms

// Hauptwerk Settings

// 61 note manuals run from 0x24 to 0x60; available MIDI codes for pistons #0-#23 (35 codes: use #1-#1C for
piston contacts) and #61-#7F (31 codes)
// In Hauptwerk use Input: Momentary piston: MIDI note-on for pistons; reversible pistons will toggle
// Initially only 8 thumb pistons wired up without diodes

// For St. Anne's Moseley
// Makin Great (MIDI Channel 02)       St. Anne's Great
// 007 Great thumb piston RED 9        0x07 Pedal trombone 16 (reversible)
// 049 Choir thumb piston C            0x08 General cancel [Wired to C thumb piston on Choir manual]
// 055 Great thumb piston 1            0x01 Great thumb piston 1
// 056 Great thumb piston 2            0x02 Great thumb piston 2
// 057 Great thumb piston 3            0x03 Great thumb piston 3
// 058 Great thumb piston 4            0x04 Great thumb piston 4
// 059 Great thumb piston 5            0x05 Great thumb piston 5
// 071 Great thumb piston GRT TO PED   0x06 Great to Pedal (reversible)

// For Madelaine Paris (All pistons reversible)
// Makin Great (MIDI Channel 02)       Madelaine I
// Great thumb piston 1                0x01 I Fonds 8
// Great thumb piston 2                0x02 I Fonds 16/8/4
// Great thumb piston 3                0x03 I Tutti
// Great thumb piston 4                0x04 Pedal 32/16
// Great thumb piston 5                0x05 Pedal Tutti
// Great thumb piston GRT TO PED       0x06 I-PED
// Choir thumb piston C                0x08 Blower [Wired to C piston on Choir manual]

//https://www.arduino.cc/en/Reference/MIDIUSB
//https://www.arduino.cc/en/Tutorial/MidiDevice
//https://github.com/arduino-libraries/MIDIUSB
//https://github.com/arduino/tutorials/blob/master/ArduinoZeroMidi/ArduinoZeroMidi.ino

#include  <MIDIUSB.h>
//#include  <pitchToFrequency.h>
//#include  <pitchToNote.h>
//#include  <frequencyToNote.h>

// Assign Arduino input/output pins to interface to Makin keyboard scanner PCB
const byte CK = A0;          // Clock output
const byte PS = A1;          // Parallel/serial select output
const byte SR = 12;          // Serial data in from 61-bit shift register (daisy chain of 4014s)
const byte ScopeSync = A5;   // Trigger oscilloscope at start of loop

// Digital inputs to read piston diode matrix columns
const byte Column1 = 0;
const byte Column2 = 1;
const byte Column3 = 2;
const byte Column4 = 3;
const byte Column5 = 4;
const byte Column6 = 5;
const byte Column7 = 6;
const byte Column8 = 7;
// Digital ouputs to drive piston diode matrix rows
const byte Row1 = 8;
const byte Row2 = 9;
const byte Row3 = 10;
const byte Row4 = 11;

// Create arrays to sample and debounce key states, true = key on, false = key off
boolean  KeyState[61];
boolean  KeyState1[61];
boolean  KeyState2[61];
boolean  PreviousKeyState;
```

```
// Create arrays to sample and debounce piston states, true = piston depressed, false = piston released
boolean PistonContactState[32]; // 18 pistons but 27 piston contacts including 9 second touch; keep array si
= 32 for code commonality across all 4 manuals
boolean  PistonContactState1[32];
boolean  PistonContactState2[32];
boolean  PreviousPistonContactState;

void setup() {

  pinMode (CK, OUTPUT);
  digitalWrite(CK, LOW);
  pinMode (PS, OUTPUT);
  digitalWrite(PS, LOW);
  pinMode (SR, INPUT_PULLUP);
  pinMode (ScopeSync, OUTPUT);
  digitalWrite(ScopeSync, LOW);
  pinMode (Column1, INPUT_PULLUP);
  pinMode (Column2, INPUT_PULLUP);
  pinMode (Column3, INPUT_PULLUP);
  pinMode (Column4, INPUT_PULLUP);
  pinMode (Column5, INPUT_PULLUP);
  pinMode (Column6, INPUT_PULLUP);
  pinMode (Column7, INPUT_PULLUP);
  pinMode (Column8, INPUT_PULLUP);
  pinMode (Row1, OUTPUT);
  pinMode (Row2, OUTPUT);
  pinMode (Row3, OUTPUT);
  pinMode (Row4, OUTPUT);
  digitalWrite(Row1, HIGH);
  digitalWrite(Row2, HIGH);
  digitalWrite(Row3, HIGH);
  digitalWrite(Row4, HIGH);
  pinMode(LED_BUILTIN, OUTPUT); // Pin 13 is LED_BUILTIN, set LED to off, use for bottom C visual indication
  digitalWrite(LED_BUILTIN, LOW);

  for (byte note = 0; note <= 60; note ++) {  // All notes off
    KeyState[note] = false;
    KeyState1[note] = false;
    KeyState2[note] = false;
  }

  for (byte PistonContact = 0; PistonContact <= 31; PistonContact ++) {  // All pistons released
    PistonContactState[PistonContact] = false;
    PistonContactState1[PistonContact] = false;
    PistonContactState2[PistonContact] = false;
  }
}

void loop() {

  // Keep code inline rather than in separate functions to keep loop period to a minimum

  digitalWrite(ScopeSync, HIGH);
  delayMicroseconds(100);
  digitalWrite(ScopeSync, LOW);

  // Interleave keyboard and piston scans to maximise time interval between scans for better contact debounc

  // First keyboard scan

  digitalWrite(PS, HIGH); // Enable parallel data entry
  delayMicroseconds(50);
  digitalWrite(CK, HIGH); // Latches data on rising edge of clock
  delayMicroseconds(50);
  digitalWrite(CK, LOW);
  digitalWrite(PS, LOW);  // Restore serial data entry
  delayMicroseconds(50);

  for (byte note = 0; note <= 60; note ++) {
    KeyState1[note] = digitalRead(SR);       // First note (bottom C) is already available without shifting,
read first then shift
```

```
    if (note < 60) {                          // 61st shift not needed
      digitalWrite(CK, HIGH);                 // Shifts data on rising edge of clock
       delayMicroseconds(20);
       digitalWrite(CK, LOW);
       delayMicroseconds(20);
    }
  }

  // First piston scan

  //  for (byte row = 0; row <= 3; row ++) {
  for (byte row = 0; row <= 1; row ++) {       // ONLY SCAN ROW 0 WHILE JUST A FEW PISTONS WIRED UP WITHOUT
DIODES

    if (row == 0) {
       digitalWrite(Row1, LOW);
       digitalWrite(Row2, HIGH);
       digitalWrite(Row3, HIGH);
       digitalWrite(Row4, HIGH);
    }

    if (row == 1) {
       digitalWrite(Row1, HIGH);
       digitalWrite(Row2, LOW);
       digitalWrite(Row3, HIGH);
       digitalWrite(Row4, HIGH);
    }

    if (row == 2) {
       digitalWrite(Row1, HIGH);
       digitalWrite(Row2, HIGH);
       digitalWrite(Row3, LOW);
       digitalWrite(Row4, HIGH);
    }

    if (row == 3) {
       digitalWrite(Row1, HIGH);
       digitalWrite(Row2, HIGH);
       digitalWrite(Row3, HIGH);
       digitalWrite(Row4, LOW);
    }

    for (byte col = 0; col <= 7; col ++) {
       byte PistonContactNumber;
       PistonContactNumber = (row * 8) + col;
        PistonContactState1[PistonContactNumber] = !digitalRead(col);   // Low = piston depressed, so invert
    }
  }

  digitalWrite(Row1, HIGH);
  digitalWrite(Row2, HIGH);
  digitalWrite(Row3, HIGH);
  digitalWrite(Row4, HIGH);

  // Second keyboard scan and process

  digitalWrite(PS, HIGH);
   delayMicroseconds(50);
  digitalWrite(CK, HIGH);
   delayMicroseconds(50);
  digitalWrite(CK, LOW);
  digitalWrite(PS, LOW);
   delayMicroseconds(50);

  for (byte note = 0; note <= 60; note ++) {
     KeyState2[note] = digitalRead(SR);
    if (note < 60) {
       digitalWrite(CK, HIGH);
       delayMicroseconds(20);
       digitalWrite(CK, LOW);
       delayMicroseconds(20);
```

```
    }
  }

  for (byte note = 0; note <= 60; note ++) {

     PreviousKeyState = KeyState[note];

     if (PreviousKeyState == false && KeyState1[note] == true && KeyState2[note] == true) {    // Note has jus
gone on
       KeyState[note] = true;
       midiEventPacket_t noteOn = {0x09, 0x91, byte(note + 0x24), 0x40}; // note + 0x24 produces an int in
compiler so use byte() to reconvert to byte value otherwise compiler outputs a narrowing conversion warning
        MidiUSB.sendMIDI(noteOn);
       MidiUSB.flush();
       if (note == 0) {
          digitalWrite(LED_BUILTIN, HIGH);    // Turn Arduino onboard LED on (Bottom C visual indication)
       }
     }

     if (PreviousKeyState == true && KeyState1[note] == false && KeyState2[note] == false) {    // Note has ju
gone off
       KeyState[note] = false;
       midiEventPacket_t noteOn = {0x08, 0x81, byte(note + 0x24), 0x40};
       MidiUSB.sendMIDI(noteOn);
       MidiUSB.flush();
       if (note == 0) {
          digitalWrite(LED_BUILTIN, LOW);    // Turn Arduino onboard LED off (Bottom C visual indication)
       }
     }
  }

  // Second piston scan and process

  //  for (byte row = 0; row <= 3; row ++) {
  for (byte row = 0; row <= 1; row ++) {       // ONLY SCAN ROW 0 WHILE JUST A FEW PISTONS WIRED UP WITHOUT
DIODES

     if (row == 0) {
       digitalWrite(Row1, LOW);
       digitalWrite(Row2, HIGH);
       digitalWrite(Row3, HIGH);
       digitalWrite(Row4, HIGH);
     }

     if (row == 1) {
       digitalWrite(Row1, HIGH);
       digitalWrite(Row2, LOW);
       digitalWrite(Row3, HIGH);
       digitalWrite(Row4, HIGH);
     }

     if (row == 2) {
       digitalWrite(Row1, HIGH);
       digitalWrite(Row2, HIGH);
       digitalWrite(Row3, LOW);
       digitalWrite(Row4, HIGH);
     }

     if (row == 3) {
       digitalWrite(Row1, HIGH);
       digitalWrite(Row2, HIGH);
       digitalWrite(Row3, HIGH);
       digitalWrite(Row4, LOW);
     }

     for (byte col = 0; col <= 7; col ++) {
       byte PistonContactNumber;
       PistonContactNumber = (row * 8) + col;
        PistonContactState2[PistonContactNumber] = !digitalRead(col);
     }
  }
```

```
  digitalWrite(Row1, HIGH);
  digitalWrite(Row2, HIGH);
  digitalWrite(Row3, HIGH);
  digitalWrite(Row4, HIGH);

  for (byte PistonContactNumber = 0; PistonContactNumber <= 31; PistonContactNumber ++) {
     PreviousPistonContactState = PistonContactState[PistonContactNumber];

    if (PreviousPistonContactState == false && PistonContactState1[PistonContactNumber] == true &&
PistonContactState2[PistonContactNumber] == true) {   // Piston has just been depressed
       PistonContactState[PistonContactNumber] = true;
       midiEventPacket_t noteOn = {0x09, 0x91, byte(PistonContactNumber + 0x01), 0x40};
       MidiUSB.sendMIDI(noteOn);
      MidiUSB.flush();
       if (PistonContactNumber == 0) {     // Turn Arduino onboard LED on (Thumb piston 1 visual indication)
         digitalWrite(LED_BUILTIN, HIGH);
      }
    }

    if (PreviousPistonContactState == true && PistonContactState1[PistonContactNumber] == false &&
PistonContactState2[PistonContactNumber] == false) {   // Piston has just been released
       PistonContactState[PistonContactNumber] = false;
       if (PistonContactNumber == 0) {    // Turn Arduino onboard LED off (Thumb piston 1 visual indication)
         digitalWrite(LED_BUILTIN, LOW);
      }
    }
  }
}
```