MIDI Keyboard Guide

Guide has been tested on

BeagleBone (target): **Debian 11.8**
PC OS (host): **Debian 11.8**

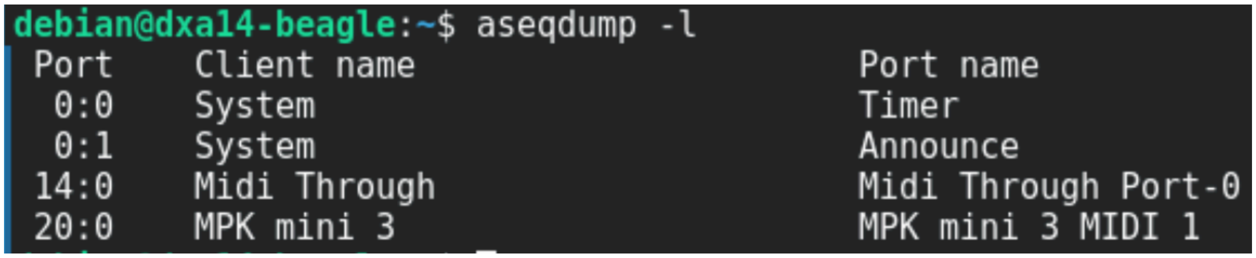
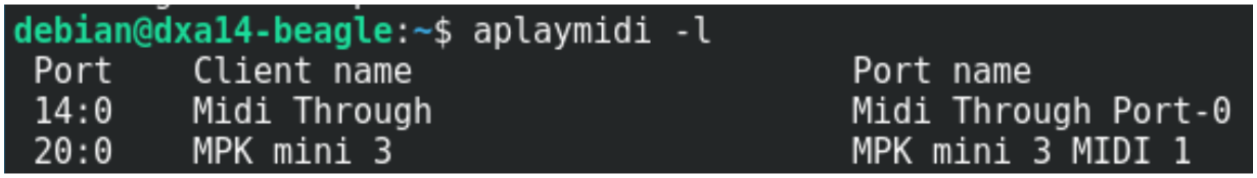
Formatting

1. Commands for the host Linux's console are show as:
   ```
   (host)$ echo "Hello PC world!"
   ```
2. Commands for the target (BeagleBone) Linux's console are shown as:
   ```
   (bbg)$ echo "Hello embedded world!"
   ```
3. Almost all commands are case sensitive.

1. Reading MIDI Keyboard Input Events: Through the Command Line

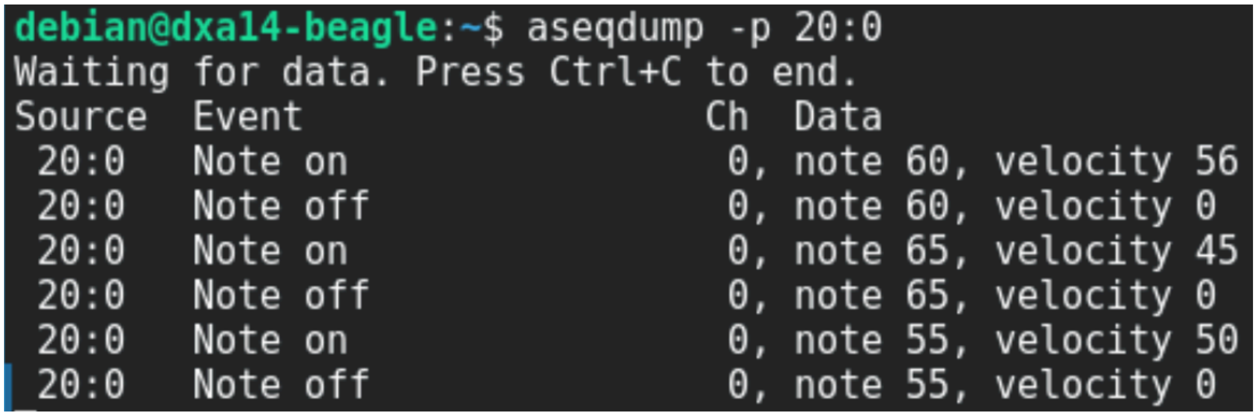
In order to read the input of a keyboard event, you must install the ALSA library on your BeagleBone. The guide for how to install the ALSA library can be found in the Zen Cape Audio Guide by Brian Fraser.[1]

Once you have completed section 1 of the Zen Cape Audio Guide, you can continue with this guide.

1. Connect your MIDI keyboard to your BBG through USB.
2. On your BBG, run the command
   ```
   (bbg)$ aseqdump -l
   ```
   This is the output that you should expect:
   
   ```
   debian@dxa14-beagle:~$ aseqdump -l
    Port    Client name                      Port name
     0:0    System                           Timer
     0:1    System                           Announce
    14:0    Midi Through                     Midi Through Port-0
    20:0    MPK mini 3                       MPK mini 3 MIDI 1
   ```
   
   ```
   debian@dxa14-beagle:~$ aplaymidi -l
    Port    Client name                      Port name
    14:0    Midi Through                     Midi Through Port-0
    20:0    MPK mini 3                       MPK mini 3 MIDI 1
   ```
   If you do not see your MIDI keyboard in the list, try to reboot your BBG or check your connection between the BBG and the keyboard.
3. Look at the port number that is listed beside your keyboard.
   - In this instance, our keyboard is the **MPK mini 3** and the port number for our keyboard is **20:0**.

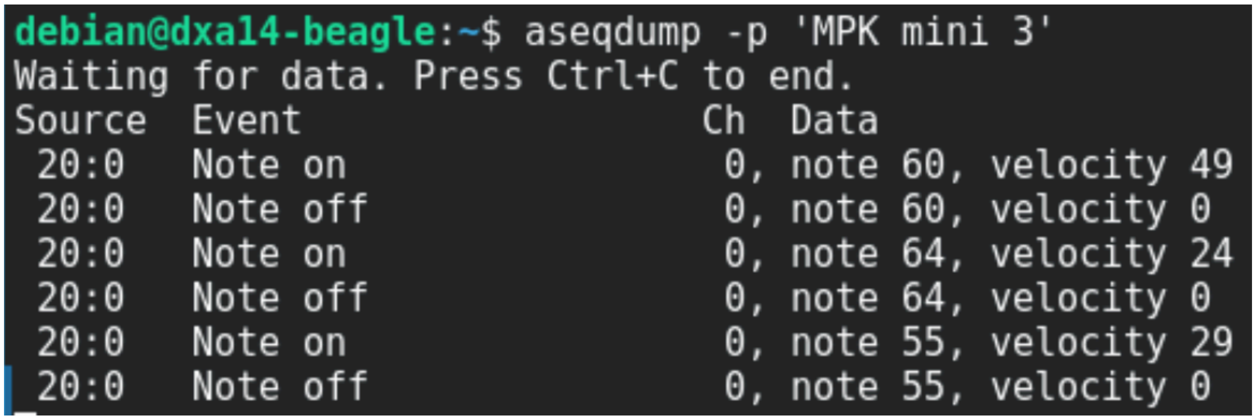
[1] https://opencoursehub.cs.sfu.ca/bfraser/grav-cms/cmpt433/guides/files/AudioGuide.pdf

4. Once you have verified your port number, you can run either one of these commands:
   (bbg)$ **aseqdump -p 20:0**



```
debian@dxa14-beagle:~$ aseqdump -p 20:0
Waiting for data. Press Ctrl+C to end.
Source  Event                  Ch  Data
 20:0   Note on                 0, note 60, velocity 56
 20:0   Note off                0, note 60, velocity 0
 20:0   Note on                 0, note 65, velocity 45
 20:0   Note off                0, note 65, velocity 0
 20:0   Note on                 0, note 55, velocity 50
 20:0   Note off                0, note 55, velocity 0
```

   (bbg)$ **aseqdump -p ‘MPK mini 3’**



```
debian@dxa14-beagle:~$ aseqdump -p 'MPK mini 3'
Waiting for data. Press Ctrl+C to end.
Source  Event                  Ch  Data
 20:0   Note on                 0, note 60, velocity 49
 20:0   Note off                0, note 60, velocity 0
 20:0   Note on                 0, note 64, velocity 24
 20:0   Note off                0, note 64, velocity 0
 20:0   Note on                 0, note 55, velocity 29
 20:0   Note off                0, note 55, velocity 0
```

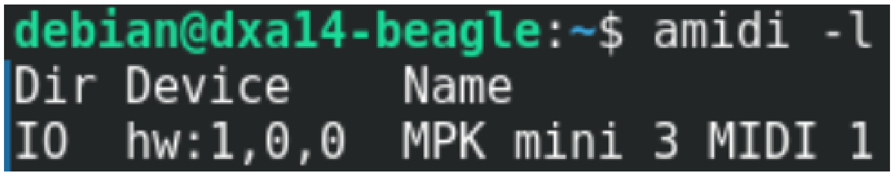
   The event shows when you play the note and when you let go of the note. The note in the data shows the actual note that the event is connected to. In this case, note 60 is equal to the note middle C on the keyboard. The note input ranges from note 0 to note 127 and the events are not limited to just keyboard notes. Other events that are made on the MIDI keyboard are also shown in this list. The velocity refers to the amount of force with which a note is played.

   You can simply terminate the program by pressing **Ctrl + C**.

2. Alternative Command: amidi

While **aseqdump** is very convenient to read, it may be quite difficult to parse the input. Another command that you can use is **amidi**. Instead of outputting words, **amidi** outputs 3 hex values.

```
(bbg)$ amidi -l
```

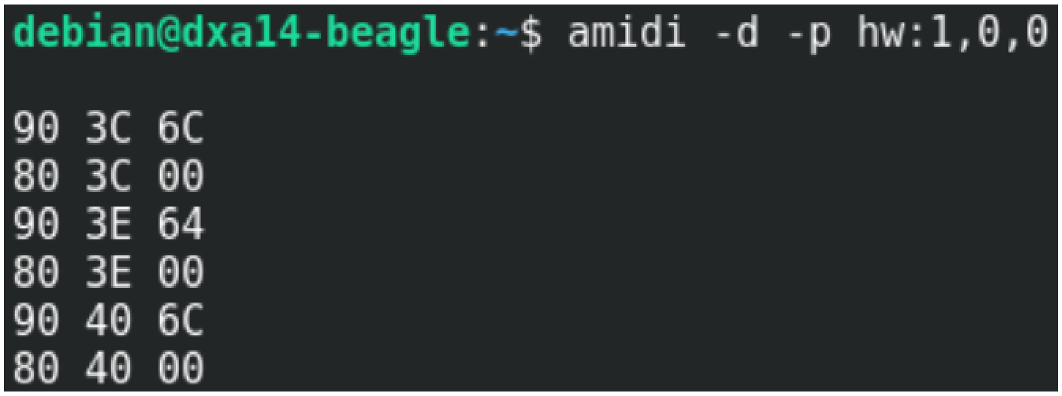


```
debian@dxa14-beagle:~$ amidi -l
Dir Device    Name
IO  hw:1,0,0  MPK mini 3 MIDI 1
```

Similar to the 'list' commands for aseqdump above, this command shows the direction of the controller, the device alias, and the device name.

To continuously dump the output, run the command below.

```
(bbg)$ amidi -d -p hw:1,0,0
```



```
debian@dxa14-beagle:~$ amidi -d -p hw:1,0,0

90 3C 6C
80 3C 00
90 3E 64
80 3E 00
90 40 6C
80 40 00
```

- The first hex value indicates the event, whether it's note on or note off. Hex value 90 means that the note is on and hex value 80 means that the note is off.
- The second hex value indicates the actual note that is played. For example, hex value 3C would be note 60.
- The third hex value refers to the velocity of the note.

3. Reading MIDI Keyboard Input Through Code

Our approach to reading MIDI keyboard input through code was by parsing the inputs from **amidi**. Some of the code here was created by consulting with chatGPT.

This function is the main function that runs the amidi command. It captures the events and prints the note played as a decimal integer.

```
void MidiReader_init(void) {
        char* midiOutput;
        printf("Press a button on the MIDI controller...\n");

        const char *command = "amidi -d -p hw:1,0,0";
        file = popen(command, "r");
        if (!file) {
            perror("popen");
            exit(1);
        }

        while(1) {
            // Execute amidi and get its output
            midiOutput = executeAmidi(file);

            // Parse MIDI output and extract note value
            int played_key = parseMIDIOutput(midiOutput);
            // Print the integer representation of note
            printf("Note Played: %s\n", played_key);

            // Wait for a moment before checking for MIDI events again
            usleep(10000);
        }

        // Free allocated memory
        free(midiOutput);
    }
```

This function parses the midi output into an int representation.

```c
// Consulted chatGPT for tokenizing output
static int parseMIDIOutput(char *midiOutput) {
   // midiOutput should look like this initially: 90 3E 31

   char *token;
   int note = -1; // Default value if note is not found

   // Gets the first part, which leads with 90
   token = strtok(midiOutput, " ");

   while(token != NULL) {
        // 90 means key is being played
        if (strcmp(token, "90") == 0) {
            // Get the next part of the output (should be the note)
            // Eg 3E
            token = strtok(NULL, " ");
            if (token != NULL) {
                // Convert hexadecimal note value to decimal
                note = (int) strtol(token, NULL, 16);
                break;
            }
        }
        // 80 means key is being released
        else if (strcmp(token, "80") == 0) {
            // Do something
            break;
        }
        token = strtok(NULL, " ");
   }
   return note;
}
```

This function executes amidi as a pipe to continuously sample the midi output

```c
// consulted chatGPT for set select functionalities
static char *executeAmidi(FILE *pipe) {
   char *output = malloc(BUFFER_SIZE);
   if (!output) {
       perror("malloc");
       exit(1);
   }
   output[0] = '\0'; // Initialize output with an empty string
   char buffer[BUFFER_SIZE];
   int i = 0;
   while(i < BUFFER_SIZE - 1) {
       fd_set set;
       struct timeval timeout;
       FD_ZERO(&set);                    // Clear the set
       FD_SET(fileno(pipe), &set);   // Add file descriptor to set
       timeout.tv_sec = 0;
       timeout.tv_usec = 10000; // 10 ms timeout between select
       int result = select(fileno(pipe) + 1, &set, NULL, NULL,
&timeout);
       if (result == -1) {
           perror("select");
           exit(1);
       } else if (result == 0) {
           break; // No data available, break the loop
       } else {
           // Data is available, read it
           char c = fgetc(pipe);
           buffer[i++] = c;
            i++
           if (c == '\n') {
               buffer[i] = '\0';
               strcat(output, buffer);
               break; // Break loop when a newline character is read
           }
       }
   }
   return output;
}
```