# MIDI

## Requirements

Table : Requirements MG

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Base ID** | **Version** | **Category** | **Item** | **Description** |
| MidG1 | GenH10 | 1.0 | Flash | Amount | The amount of Flash memory is limited.  *Rationale: Cost perspective.* |

## Hardware

### Requirements

Table : Requirements MH

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Base ID** | **Version** | **Category** | **Item** | **Description** |
| MidH100 | - | 1.0 | Connectors | Inputs | Multiple MIDI In connectors should be supported.  *Rationale: at least one for a main keyboard, secondary keyboard and foot controller.* |
| MidH110 | - | 1.0 | Connectors | Outputs | Multiple MIDI Out connectors should be supported.  *Rationale: Possibility to send MDI messages to multiple keyboards without the need to use another THRU connector.* |
| MidH120 | - | 1.0 | Connectors | Thrus | Each MIDI In connector should have its MIDI Thru connector. |

### Inputs/Outputs

TODO

### Design

#### Safety

Because of REQ GenH30, connected devices should not damage the device. Therefore, opto isolators will be used for the MIDI In connectors.

#### Opto Isolators Types

The following are options which have been checked:

1. 6N137: Default opto couplers, however, these are not fast enough for MIDI messages
2. 6N138: Faster, however, not selected since H1L11 opto isolators are adviced (see below); advised by MIDI protocol.
3. H11L1: Faster, see Arduino-stack exchange links <https://arduino.stackexchange.com/questions/39908/h11l1-opto-isolator-does-not-pass-signal-to-arduino-correctly> and <https://electronics.stackexchange.com/questions/311889/i-cannot-get-opto-coupler-h11l1-to-work-as-midi-input> ).
4. PC900: advised by MIDI protocol; not checked, since I already have 6N138s and H11L1s.
5. Multi channel opto isolators: since multiple opto isolators are needed (one for each MIDI channels, 3 in total), a three channel opto isolator would be ideal. However, there are dual opto isolators, but these are too slow for MIDI messages.

The selection is made for H11L1, although not as one of the advised opto isolators. These are used in MIDI circuits however, and work flawlessly. Also since these ICs are only DIP6 instead of DIP8 and need fewer components around (resistors/5V lines), H11L1 is the best solution. Also the resistors around the H11L1 can all be 220 ohm like in the advised MIDI circuit.

#### EMI/EMC performance

Since in a life situation EMI can result in problems, for all MIDI inputs/outputs inductor beads will be used. The advised are not to be found for a cheap price, however a test will be made with 3T 6\*10 R6H (six hole) magnetic beads which have similar stats (TODO) as the advised inductors from the MIDI protocol.

#### Amount of MIDI Connectors

Every MIDI In uses a Serial (RX). Tests using SoftwareSerial results in unreliable MIDI messages. Since this can result Note Off commands being corrupted/not received, this is fully unacceptable in a life situation. This happens only when lots of messages are received (e.g. during lots of CC messages like the modulation wheel/after touch). Therefore, SoftwareSerial is unusable for MIDI.

That means each MIDI In needs a hardware RX. Since the Arduino Mega is the only Arduino (working on 5V), giving 4 UARTs, an Arduino Mega is necessary. One UART possibly has to be used for the RF communication, leaving 3 MIDI Ins available.

Therefore, it is logical to have 3 MIDI Thrus and 3 MIDI Outs.

This satisfies REQ MidH110, MidH120 and MidH130.

#### Diodes Type

For the MIDI Ins, diodes are used. The most default used are 1N4001 diodes, which work (tested), however 1N4148 seem to be better suitable for fast performance, so 1N4148 diodes will be used.

#### Diagnostics LEDs

Table : MIDI Diagnostics LEDs

|  |  |  |
| --- | --- | --- |
| **Function** | **LED Color** | **Description** |
| Power | Blue (generic) | Off: Power off  On: Power on |
| RF | Yellow (generic) | Off: empty message transmitting/receiving  Slow blinking: contact with controller  Double fast blinking per second: no contact with slave  Triple fast blinking per second: problem with RF  On: non empty message transmitting/receiving |
| MIDI In | Green | Off: Not receiving MIDI data  On: Receiving MIDI data |
| MIDI Out | Green/Red (bicolor) | Off: Not transmitting MIDI data  Green: Transmitting unprocessed MIDI data  Red: Transmitting processed MIDI data |

Note that if the GUI Device shows errors whenever possible.

### Breadboard Layout

For the MIDI device, two breadboards are needed, since there fit only about 6 MIDI connectors on one breadboard. Since 9 MIDI connectors will be placed, the breadboards will contain the following items:

1. Breadboard 1 (830 holes breadboard)
   1. MIDI IN 1, MIDI OUT 1, MIDI THRU 1
   2. MIDI IN 2, MIDI OUT 2, MIDI THRU 2
2. Breadboard 2 (400 holes breadboard or half 830 holes breadboard)
   1. MIDI IN 3, MIDI OUT 3, MIDI THRU 3

### Proto Layout

TODO

### Component List

* Enclosure
* Arduino (compatible) Mega
* SD Data Logger Shield (with SRam)
* Arduino Uno Protype Shield, with optocouplers and other logic.
* 5 MIDI DIN female connectors
* 4 dual green/red 3mm LEDs (for MIDI ins and MIDI out)
* 1 green 3mm LED (for MIDI thru)
* 12V adapter
* 6 x 220 Ohm resistors
* TODO resistors
* Bypass capacitors: TODO
* 2 x 4N189 diodes
* 3 x 6N137 Optocouplers

## Software

### Requirements

Table : Requirements MS

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Base ID** | **Version** | **Category** | **Item** | **Description** |
| MidS100 | TODO | 1.0 | Messages | Prevent unnecessary | A mechanism to immediately return/forward messages should be implemented, and to not send messages to the Controller.  *Rationale: To prevent unnecessary messages (costing time/bandwidth).* |
|  |  |  |  |  |  |

### Design

#### MS100: Preventing unwanted messages

To prevent unwanted messages, lists are needed for what messages:

* will be returned/handled immediately by this device
* will be send to the Controller.

For both, one bit is needed.

Assuming 16 MIDI channels, 128 notes, this results in 16 \* 128 \* 2 bits = 512 bytes

For CC, there are 16 MIDI channels, 128 CCs, resulting in the same number of bytes.

This totals 1024 bytes, which is 1 KB. Since the Arduino Mega has 4 KB, this suffices.

### Memory Usage

#### Trigger conditions

All trigger conditions need to be stored. MIDI trigger conditions can be per MIDI channel, note or CC, its velocity/release velocity/value and per type (change, into a region, out of a region).

There can be a maximum of 64 MIDI channels (4 MIDI In’s with each 16 MIDI channels).

There can be 128 note or CC values.

There also can be many trigger regions, even for the same note or CC.

This would result in an endless number of trigger conditions. Therefore, a more flexible way is used:

* Per MIDI channel a table is created, containing:
  + A list which is generic for all notes.
  + A list per note
  + A list per CC
  + A list for all program changes
* Each condition also has a type (value change, out of a region, inside a region). Therefore the current values should be stored to be compared against.
* Only USED midi channels, USED notes/CCs and USED program changes are present in the list
* Trigger conditions for note velocities/release velocities and CC values are not possible; these will be handled by filtering the property inside command.
* The application (APP) will make sure that the maximum amount of memory can be used, by assigning all lists dynamically.

#### Note On/Off

Assuming 4 MIDI channels are used, there will be a list generically for all notes. This list can have multiple entries, for the entire range, or one or more specific ranges.

*Example: A Note On command for note C4 with velocity 80 on MiDI channel 2 is received. First the generic list for MIDI channel 2 is checked. All conditions within the list are compared and when a trigger matches, a message is sent to the Controller. Than all conditions for the specified note is checked against all conditions.*

The memory needed will be for e.g. 4 MIDI Channels, 1 generic list (per MIDI Channel) and assuming 50 note (including region) will be:

| **Category** | **Item** | **Size** | **Unit** | **For Memory Counting** | **Remarks** |
| --- | --- | --- | --- | --- | --- |
| General | Used MIDI channels | 64 | MIDI channels |  | Address, 0 if not used. |
|  | Bytes for used MIDI channels | 128 | MIDI channels | 128 |  |
| Notes | Used MIDI channels | 4 | MIDI channels |  |  |
|  | Notes per MIDI channel | 129 | Notes |  | Address, 128 notes + 1 generic |
| CCs | CC per MIDI channel | 129 | CCs |  | As above |
| Lists | Total tables | 1024 | Tables |  |  |
| Table | Assumed length | 30 | Items |  | 30 items |
|  | Item size | 3 | Bytes |  | Assuming 3 types (1 byte), start region value (1 byte), end region value (1 byte) |
|  | Table size | 90 | Bytes |  |  |
|  | Tables size | 92,160 | Bytes | 92,160 |  |

### Timing Performance

MS100 will reduce the number of messages.

Worst case MIDI latency is defined by the sum of:

1. Time to receive a MIDI message
2. Time to process it by the device
3. Time to send it to the Controller
4. Time to process it by the Controller
5. Time to receive a message by the Controller
6. Time to send it through MIDI.

For a typical message of 3 bytes, this results in:

1. MIDI has a baudrate of 31,250 baud, which results approximately in a receive/time of 0,768 ms
2. TODO
3. This will be calculated in TODO.
4. See 3
5. See 2
6. See 1

This totals to 0,768 ms + TODO + 0,768 ms = TODO.

## Testing

### Unit Tests

TODO

### Integration Tests

TODO

# Proximity

## Generic

### Requirements

TODO

## Hardware

### Requirements

TODO

### Inputs/Outputs

TODO

### Design

#### Proximity LEDs

Table : Audio Diagnostics LEDs

|  |  |  |
| --- | --- | --- |
| **Function** | **LED Color** | **Description** |
| Power | Blue (generic) | Off: Power off  On: Power on |
| RF | Yellow (generic) | Off: empty message transmitting/receiving  Slow blinking: contact with controller  Double fast blinking per second: no contact with slave  Triple fast blinking per second: problem with RF  On: non empty message transmitting/receiving |

Note that if the GUI Device shows errors whenever possible.

### Breadboard Layout

TODO

### Proto Layout

TODO

### Component List

TODO

## Software

### Requirements

TODO

### Design

TODO

### Memory Usage

This device does not receive messages, since there are no outputs.

Only a buffer is needed for sending received signals from switches and inputs, which is TODO bytes.

### Timing Performance

Only the translation from signals to messages is needed, and sending them to the Controller:

TODO

## Testing

### Unit Tests

TODO

### Integration Tests

TODO

# Pedals/Switches

## Generic

### Requirements

TODO

## Hardware

### Requirements

TODO

### Inputs/Outputs

TODO

### Design

#### Diagnostics LEDs

Table : Pedals/switches Diagnostics LEDs

|  |  |  |
| --- | --- | --- |
| **Function** | **LED Color** | **Description** |
| Power | Blue (generic) | Off: Power off  On: Power on |
| RF | Yellow (generic) | Off: empty message transmitting/receiving  Slow blinking: contact with controller  Double fast blinking per second: no contact with slave  Triple fast blinking per second: problem with RF  On: non empty message transmitting/receiving |

Note that if the GUI Device shows errors whenever possible.

### Breadboard Layout

TODO

### Proto Layout

TODO

### Component List

TODO

## Software

### Requirements

TODO

### Design

TODO

### Memory Usage

This device does not receive messages, since there are no outputs.

Only a buffer is needed for sending received signals from switches and inputs, which is TODO bytes.

### Timing Performance

Only the translation from signals to messages is needed, and sending them to the Controller:

TODO

## Testing

### Unit Tests

TODO

### Integration Tests

TODO

# Remote

## Generic

### Requirements

TODO

## Hardware

### Requirements

TODO

### Inputs/Outputs

TODO

### Design

#### Diagnostics LEDs

Table : Remote Diagnostics LEDs

|  |  |  |
| --- | --- | --- |
| **Function** | **LED Color** | **Description** |
| Power | Blue (generic) | Off: Power off  On: Power on |
| RF | Yellow (generic) | Off: empty message transmitting/receiving  Slow blinking: contact with controller  Double fast blinking per second: no contact with slave  Triple fast blinking per second: problem with RF  On: non empty message transmitting/receiving |

Note that if the GUI Device shows errors whenever possible.

### Breadboard Layout

TODO

### Proto Layout

TODO

### Component List

TODO

## Software

### Requirements

TODO

### Design

TODO

### Memory Usage

This device does not receive messages, since there are no outputs.

Only a buffer is needed for sending received signals from pressed/changed controls, which is TODO bytes.

### Timing Performance

Only the translation from signals to messages is needed, and sending them to the Controller:

TODO

## Testing

### Unit Tests

TODO

### Integration Tests

TODO

# USB MIDI

## Generic

### Requirements

TODO

## Hardware

### Requirements

TODO

### Inputs/Outputs

TODO

### Design

#### Diagnostics LEDs

Table :USB MIDI Diagnostics LEDs

|  |  |  |
| --- | --- | --- |
| **Function** | **LED Color** | **Description** |
| Power | Blue (generic) | Off: Power off  On: Power on |
| RF | Yellow (generic) | Off: empty message transmitting/receiving  Slow blinking: contact with controller  Double fast blinking per second: no contact with slave  Triple fast blinking per second: problem with RF  On: non empty message transmitting/receiving |

Note that if the GUI Device shows errors whenever possible.

### Breadboard Layout

TODO

### Proto Layout

TODO

### Component List

TODO

## Software

### Requirements

TODO

### Design

TODO

### Memory Usage

TODO

### Timing Performance

TODO

## Testing

### Unit Tests

TODO

### Integration Tests

TODO

# Debug

## Generic

### Requirements

Table : : Requirements EG

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Base ID** | **Version** | **Category** | **Item** | **Description** |
| DbgG1 |  | 1.0 | Debug | Devices | Debugging from all devices will be possible, without using the normal communication (like RF).  *Rationale: using the normal communication might interfere the normal messages sent over it.* |
| DbgG2 |  | 1.0 | Debug | Amount | All devices can send debug messages to the Debug device.  *Rationale: Debugging should be possible over multiple devices.* |
| DbgG3 |  | 1.0 | Debug | Amount | The hardware RX pin should only be used for sending debug messages to the computer showing diagnostics messages.  *Rationale: Using the hardware RX pin for other purposes, would corrupt the messages.* |

## Hardware

### Requirements

TODO

### Inputs/Outputs

TODO

### Design

#### Software Serial pin layout

Since only one directional communication is needed, the following pin layout will be used:

* Pin 0 (RX): For hardware debug port/serial
* Pin 1 (TX): For hardware debug port/serial
* Pin 2 .. 9: 8 software serial RX pins for 8 devices
* Pin 10..13 can be used for LEDs (TODO)

#### Diagnostics LEDs

Table : Debug Diagnostics LEDs

|  |  |  |
| --- | --- | --- |
| **Function** | **LED Color** | **Description** |
| Power | Blue (generic) | Off: Power off  On: Power on |
| RF | Yellow (generic) | Off: empty message transmitting/receiving  Slow blinking: contact with controller  Double fast blinking per second: no contact with slave  Triple fast blinking per second: problem with RF  On: non-empty message transmitting/receiving |

Note that if the GUI Device shows errors whenever possible.

### Breadboard Layout

The breadboard contains the LEDs as described in paragraph **Error! Reference source not found.**. This will take up a 400 holes breadboard or half 830 breadboard.

### Proto Layout

TODO

### Component List

TODO

## Software

### Requirements

### Design

#### Arduino Type

Since only one hardware RX is needed, and the functionality is rather simple, an Arduino Uno will provide enough.

#### Software serials

To meet REQ DbgG3, for each device a Software Serial will be used. 8 can be used to debug 8 different devices. A loop will be made to iterate over all 8 software serial RX’s, and print it to the hardware serial.

### Memory Usage

Each software serial buffer uses 64 bytes, the hardware serial too, this means (8 + 1) \* 64 = 576 bytes for buffers.

### Timing Performance

When a lot of messages will be received, buffers can be full, however the debug messages are not critical.

## Testing

### Unit Tests

TODO

### Integration Tests

TODO

Appendix A: New device template

# New Device Template

## Generic

### Requirements

TODO

## Hardware

### Requirements

TODO

### Inputs/Outputs

TODO

### Design

#### Diagnostics LEDs

Table : NEW\_DEVICE Diagnostics LEDs

|  |  |  |
| --- | --- | --- |
| **Function** | **LED Color** | **Description** |
| Power | Blue (generic) | Off: Power off  On: Power on |
| RF | Yellow (generic) | Off: empty message transmitting/receiving  Slow blinking: contact with controller  Double fast blinking per second: no contact with slave  Triple fast blinking per second: problem with RF  On: non empty message transmitting/receiving |

Note that if the GUI Device shows errors whenever possible.

### Breadboard Layout

TODO

### Proto Layout

TODO

### Component List

TODO

## Software

### Requirements

TODO

### Design

TODO

### Memory Usage

TODO

### Timing Performance

TODO

## Testing

### Unit Tests

TODO

### Integration Tests

TODO