Mestra

Version: 0.1

Date: Aug 8, 2017

Author: Michel Keijzers

# Open Issues / Early TODOs

Table : Open Issues

|  |  |
| --- | --- |
| **Issue Nr** | **Description** |
|  | Stop all notes command (per MC?) / Panic message |
|  | Trigger: Flank up/flank down |
|  | Trigger in range? |
|  | Trigger combine (boolean or-ing of multiple triggers)? |

# History

Table : History

|  |  |
| --- | --- |
| **Date/period** | **Actions** |
| 20 Feb 2017 | Started reading general info, created document |
| 29 Mar 2017 | Decided for SD copy to SRAM, Arduino Mega. |
| 6 June 2017 | Reformatted texts |
| 1 July 2017 | Double RF idea |
| 8 Aug 2017 | RF tests |

# Abbreviations

Table : Abbreviations

|  |  |
| --- | --- |
| **Abbreviation** | **Description** |
| CC | Continuous Controller |
| DMX | Digital MultipleXed) |
| Mestra | Message Transformmer |
| MIDI | Musical Instrument Digital Interface |
| UI | User Interface |

# Glossary

Table : Glossary

|  |  |
| --- | --- |
| **Term** | **Description** |
| Aftertouch | A type of CC when further pressing a physical note on a (MIDI) keyboard after it has been pressed already. |
| CC | A controller used on keyboards/workstations. |
| Command | A specific instruction to change a property or send a MIDI message. |
| Controller | This device controls all slaves. |
| DMX | Protocol/connection for light systems. |
| Input Device | These devices are slaves and get input by devices outside the Mestra system, like MIDI signals by pedals/switches etc. |
| Input/Output Device | These devices are slaves that both can receive or send messages outside the Mestra system. |
| Joystick | Device on a MIDI device that can change the property of a sound (like modulation or pitch). |
| MIDI | Music instrument protocol |
| MIDI Channel | One of the 16 MIDI connections supported by MIDI to connect music instruments. |
| MIDI Message | A message using the MIDI protocol |
| Note On | A MIDI message that switches a key on. |
| Note Off | A MIDI message that switches a key off. |
| Output Device | These devices are slaves that outputs messages outside the Mestra system, like MIDI or DMX data. |
| Pitch | The frequency of a note. |
| Property | A property of a MIDI in message that used to change and later send by a MIDI message. |
| Release Velocity | Property of the NoteOff MIDI message for the speed a key has been released. |
| Remote Device | Device to remotely send messages (to the Controller). |
| RF | Radio Frequency (module) |
| Rule | A rule is the execution of one or more commands when a trigger is activated. |
| Slave | A device that is controlled by the Controller. These devices are generally input and/or output devices. |
| Trigger | A received MIDI message that will cause a rule to be executed. |
| UI (device) | Device for showing output and entering input. |
| Velocity | Property of the NoteOn MIDI message for the speed a key has been pressed. |

# Useful Links

Table : Useful Links

|  |  |  |
| --- | --- | --- |
| **Category** | **Description** | **Link** |
|  |  |  |

# Table of Contents

Contents

[Open Issues / Early TODOs 2](#_Toc489634921)

[History 2](#_Toc489634922)

[Abbreviations 2](#_Toc489634923)

[Glossary 2](#_Toc489634924)

[Useful Links 3](#_Toc489634925)

[Table of Contents 4](#_Toc489634926)

[List of Tables 13](#_Toc489634927)

[List of Figures 14](#_Toc489634928)

[1 Introduction 15](#_Toc489634929)

[1.1 System Boundaries 15](#_Toc489634930)

[1.2 One or more Arduinos 15](#_Toc489634931)

[1.3 Devices 15](#_Toc489634932)

[1.3.1 Device Descriptions 16](#_Toc489634933)

[1.4 Chapter Division 16](#_Toc489634934)

[2 Generic 18](#_Toc489634935)

[2.1 AP, Hardware and Software 18](#_Toc489634936)

[2.1.1 Requirements 18](#_Toc489634937)

[2.1.2 Design 18](#_Toc489634938)

[2.2 Generic Hardware 18](#_Toc489634939)

[2.2.1 Requirements 18](#_Toc489634940)

[2.2.2 Design 19](#_Toc489634941)

[2.3 Generic Software 25](#_Toc489634942)

[2.3.1 Requirements 25](#_Toc489634943)

[2.3.2 Design 27](#_Toc489634944)

[2.4 Testing 28](#_Toc489634945)

[2.4.1 Unit Tests 28](#_Toc489634946)

[2.4.2 Integration Tests 28](#_Toc489634947)

[3 Application (AP) 29](#_Toc489634948)

[3.1 Requirements 29](#_Toc489634949)

[3.2 Design 29](#_Toc489634950)

[3.2.1 Programming Language 29](#_Toc489634951)

[3.3 Testing 29](#_Toc489634952)

[3.3.1 Unit Tests 29](#_Toc489634953)

[3.3.2 Integration Tests 29](#_Toc489634954)

[4 Controller 30](#_Toc489634955)

[4.1 Generic 30](#_Toc489634956)

[4.1.1 Requirements 30](#_Toc489634957)

[4.2 Hardware 30](#_Toc489634958)

[4.2.1 Requirements 30](#_Toc489634959)

[4.2.2 Inputs/Outputs 30](#_Toc489634960)

[4.2.3 Design 30](#_Toc489634961)

[4.2.4 Breadboard Layout 32](#_Toc489634962)

[4.2.5 Proto Layout 32](#_Toc489634963)

[4.2.6 Component List 33](#_Toc489634964)

[4.3 Software 33](#_Toc489634965)

[4.3.1 Requirements 33](#_Toc489634966)

[4.3.2 Design 33](#_Toc489634967)

[4.3.3 Memory Usage 33](#_Toc489634968)

[4.3.4 Timing Performance 34](#_Toc489634969)

[4.4 Testing 34](#_Toc489634970)

[4.4.1 Unit Tests 34](#_Toc489634971)

[4.4.2 Integration Tests 34](#_Toc489634972)

[5 Audio 36](#_Toc489634973)

[5.1 Generic 36](#_Toc489634974)

[5.1.1 Requirements 36](#_Toc489634975)

[5.2 Hardware 36](#_Toc489634976)

[5.2.1 Requirements 36](#_Toc489634977)

[5.2.2 Inputs/Outputs 36](#_Toc489634978)

[5.2.3 Design 36](#_Toc489634979)

[5.2.4 Breadboard Layout 36](#_Toc489634980)

[5.2.5 Proto Layout 36](#_Toc489634981)

[5.2.6 Component List 36](#_Toc489634982)

[5.3 Software 36](#_Toc489634983)

[5.3.1 Requirements 36](#_Toc489634984)

[5.3.2 Design 36](#_Toc489634985)

[5.3.3 Memory Usage 36](#_Toc489634986)

[5.3.4 Timing Performance 36](#_Toc489634987)

[5.4 Testing 37](#_Toc489634988)

[5.4.1 Unit Tests 37](#_Toc489634989)

[5.4.2 Integration Tests 37](#_Toc489634990)

[6 DMX 38](#_Toc489634991)

[6.1 Generic 38](#_Toc489634992)

[6.1.1 Requirements 38](#_Toc489634993)

[6.2 Hardware 38](#_Toc489634994)

[6.2.1 Requirements 38](#_Toc489634995)

[6.2.2 Inputs/Outputs 38](#_Toc489634996)

[6.2.3 Design 38](#_Toc489634997)

[6.2.4 Breadboard Layout 38](#_Toc489634998)

[6.2.5 Proto Layout 38](#_Toc489634999)

[6.2.6 Component List 38](#_Toc489635000)

[6.3 Software 38](#_Toc489635001)

[6.3.1 Requirements 38](#_Toc489635002)

[6.3.2 Design 38](#_Toc489635003)

[6.3.3 Memory Usage 38](#_Toc489635004)

[6.3.4 Timing Performance 38](#_Toc489635005)

[6.4 Testing 39](#_Toc489635006)

[6.4.1 Unit Tests 39](#_Toc489635007)

[6.4.2 Integration Tests 39](#_Toc489635008)

[7 Drum Pad 40](#_Toc489635009)

[7.1 Generic 40](#_Toc489635010)

[7.1.1 Requirements 40](#_Toc489635011)

[7.2 Hardware 40](#_Toc489635012)

[7.2.1 Requirements 40](#_Toc489635013)

[7.2.2 Inputs/Outputs 40](#_Toc489635014)

[7.2.3 Design 40](#_Toc489635015)

[7.2.4 Breadboard Layout 40](#_Toc489635016)

[7.2.5 Proto Layout 40](#_Toc489635017)

[7.2.6 Component List 40](#_Toc489635018)

[7.3 Software 40](#_Toc489635019)

[7.3.1 Requirements 40](#_Toc489635020)

[7.3.2 Design 40](#_Toc489635021)

[7.3.3 Memory Usage 40](#_Toc489635022)

[7.3.4 Timing Performance 40](#_Toc489635023)

[7.4 Testing 41](#_Toc489635024)

[7.4.1 Unit Tests 41](#_Toc489635025)

[7.4.2 Integration Tests 41](#_Toc489635026)

[8 Drums Trigger 42](#_Toc489635027)

[8.1 Generic 42](#_Toc489635028)

[8.1.1 Requirements 42](#_Toc489635029)

[8.2 Hardware 42](#_Toc489635030)

[8.2.1 Requirements 42](#_Toc489635031)

[8.2.2 Inputs/Outputs 42](#_Toc489635032)

[8.2.3 Design 42](#_Toc489635033)

[8.2.4 Breadboard Layout 42](#_Toc489635034)

[8.2.5 Proto Layout 42](#_Toc489635035)

[8.2.6 Component List 42](#_Toc489635036)

[8.3 Software 42](#_Toc489635037)

[8.3.1 Requirements 42](#_Toc489635038)

[8.3.2 Design 42](#_Toc489635039)

[8.3.3 Memory Usage 42](#_Toc489635040)

[8.3.4 Timing Performance 42](#_Toc489635041)

[8.4 Testing 43](#_Toc489635042)

[8.4.1 Unit Tests 43](#_Toc489635043)

[8.4.2 Integration Tests 43](#_Toc489635044)

[9 GUI 44](#_Toc489635045)

[9.1 Generic 44](#_Toc489635046)

[9.1.1 Requirements 44](#_Toc489635047)

[9.2 Hardware 44](#_Toc489635048)

[9.2.1 Requirements 44](#_Toc489635049)

[9.2.2 Inputs/Outputs 44](#_Toc489635050)

[9.2.3 Design 44](#_Toc489635051)

[9.2.4 Breadboard Layout 44](#_Toc489635052)

[9.2.5 Proto Layout 44](#_Toc489635053)

[9.2.6 Component List 44](#_Toc489635054)

[9.3 Software 44](#_Toc489635055)

[9.3.1 Requirements 44](#_Toc489635056)

[9.3.2 Design 44](#_Toc489635057)

[9.3.3 Memory Usage 44](#_Toc489635058)

[9.3.4 Timing Performance 44](#_Toc489635059)

[9.4 Testing 45](#_Toc489635060)

[9.4.1 Unit Tests 45](#_Toc489635061)

[9.4.2 Integration Tests 45](#_Toc489635062)

[10 Microphones 46](#_Toc489635063)

[10.1 Generic 46](#_Toc489635064)

[10.1.1 Requirements 46](#_Toc489635065)

[10.2 Hardware 46](#_Toc489635066)

[10.2.1 Requirements 46](#_Toc489635067)

[10.2.2 Inputs/Outputs 46](#_Toc489635068)

[10.2.3 Design 46](#_Toc489635069)

[10.2.4 Breadboard Layout 46](#_Toc489635070)

[10.2.5 Proto Layout 46](#_Toc489635071)

[10.2.6 Component List 46](#_Toc489635072)

[10.3 Software 46](#_Toc489635073)

[10.3.1 Requirements 46](#_Toc489635074)

[10.3.2 Design 46](#_Toc489635075)

[10.3.3 Memory Usage 46](#_Toc489635076)

[10.3.4 Timing Performance 47](#_Toc489635077)

[10.4 Testing 47](#_Toc489635078)

[10.4.1 Unit Tests 47](#_Toc489635079)

[10.4.2 Integration Tests 47](#_Toc489635080)

[11 MIDI 48](#_Toc489635081)

[11.1 Requirements 48](#_Toc489635082)

[11.2 Hardware 48](#_Toc489635083)

[11.2.1 Requirements 48](#_Toc489635084)

[11.2.2 Inputs/Outputs 48](#_Toc489635085)

[11.2.3 Design 48](#_Toc489635086)

[11.2.4 Breadboard Layout 50](#_Toc489635087)

[11.2.5 Proto Layout 50](#_Toc489635088)

[11.2.6 Component List 50](#_Toc489635089)

[11.3 Software 50](#_Toc489635090)

[11.3.1 Requirements 50](#_Toc489635091)

[11.3.2 Design 50](#_Toc489635092)

[11.3.3 Memory Usage 51](#_Toc489635093)

[11.3.4 Timing Performance 52](#_Toc489635094)

[11.4 Testing 52](#_Toc489635095)

[11.4.1 Unit Tests 52](#_Toc489635096)

[11.4.2 Integration Tests 52](#_Toc489635097)

[12 Proximity 54](#_Toc489635098)

[12.1 Generic 54](#_Toc489635099)

[12.1.1 Requirements 54](#_Toc489635100)

[12.2 Hardware 54](#_Toc489635101)

[12.2.1 Requirements 54](#_Toc489635102)

[12.2.2 Inputs/Outputs 54](#_Toc489635103)

[12.2.3 Design 54](#_Toc489635104)

[12.2.4 Breadboard Layout 54](#_Toc489635105)

[12.2.5 Proto Layout 54](#_Toc489635106)

[12.2.6 Component List 54](#_Toc489635107)

[12.3 Software 54](#_Toc489635108)

[12.3.1 Requirements 54](#_Toc489635109)

[12.3.2 Design 54](#_Toc489635110)

[12.3.3 Memory Usage 54](#_Toc489635111)

[12.3.4 Timing Performance 55](#_Toc489635112)

[12.4 Testing 55](#_Toc489635113)

[12.4.1 Unit Tests 55](#_Toc489635114)

[12.4.2 Integration Tests 55](#_Toc489635115)

[13 Pedals/Switches 56](#_Toc489635116)

[13.1 Generic 56](#_Toc489635117)

[13.1.1 Requirements 56](#_Toc489635118)

[13.2 Hardware 56](#_Toc489635119)

[13.2.1 Requirements 56](#_Toc489635120)

[13.2.2 Inputs/Outputs 56](#_Toc489635121)

[13.2.3 Design 56](#_Toc489635122)

[13.2.4 Breadboard Layout 56](#_Toc489635123)

[13.2.5 Proto Layout 56](#_Toc489635124)

[13.2.6 Component List 56](#_Toc489635125)

[13.3 Software 56](#_Toc489635126)

[13.3.1 Requirements 56](#_Toc489635127)

[13.3.2 Design 56](#_Toc489635128)

[13.3.3 Memory Usage 56](#_Toc489635129)

[13.3.4 Timing Performance 57](#_Toc489635130)

[13.4 Testing 57](#_Toc489635131)

[13.4.1 Unit Tests 57](#_Toc489635132)

[13.4.2 Integration Tests 57](#_Toc489635133)

[14 Remote 58](#_Toc489635134)

[14.1 Generic 58](#_Toc489635135)

[14.1.1 Requirements 58](#_Toc489635136)

[14.2 Hardware 58](#_Toc489635137)

[14.2.1 Requirements 58](#_Toc489635138)

[14.2.2 Inputs/Outputs 58](#_Toc489635139)

[14.2.3 Design 58](#_Toc489635140)

[14.2.4 Breadboard Layout 58](#_Toc489635141)

[14.2.5 Proto Layout 58](#_Toc489635142)

[14.2.6 Component List 58](#_Toc489635143)

[14.3 Software 58](#_Toc489635144)

[14.3.1 Requirements 58](#_Toc489635145)

[14.3.2 Design 58](#_Toc489635146)

[14.3.3 Memory Usage 58](#_Toc489635147)

[14.3.4 Timing Performance 59](#_Toc489635148)

[14.4 Testing 59](#_Toc489635149)

[14.4.1 Unit Tests 59](#_Toc489635150)

[14.4.2 Integration Tests 59](#_Toc489635151)

[15 USB MIDI 60](#_Toc489635152)

[15.1 Generic 60](#_Toc489635153)

[15.1.1 Requirements 60](#_Toc489635154)

[15.2 Hardware 60](#_Toc489635155)

[15.2.1 Requirements 60](#_Toc489635156)

[15.2.2 Inputs/Outputs 60](#_Toc489635157)

[15.2.3 Design 60](#_Toc489635158)

[15.2.4 Breadboard Layout 60](#_Toc489635159)

[15.2.5 Proto Layout 60](#_Toc489635160)

[15.2.6 Component List 60](#_Toc489635161)

[15.3 Software 60](#_Toc489635162)

[15.3.1 Requirements 60](#_Toc489635163)

[15.3.2 Design 60](#_Toc489635164)

[15.3.3 Memory Usage 60](#_Toc489635165)

[15.3.4 Timing Performance 60](#_Toc489635166)

[15.4 Testing 61](#_Toc489635167)

[15.4.1 Unit Tests 61](#_Toc489635168)

[15.4.2 Integration Tests 61](#_Toc489635169)

[16 Debug 62](#_Toc489635170)

[16.1 Generic 62](#_Toc489635171)

[16.1.1 Requirements 62](#_Toc489635172)

[16.2 Hardware 62](#_Toc489635173)

[16.2.1 Requirements 62](#_Toc489635174)

[16.2.2 Inputs/Outputs 62](#_Toc489635175)

[16.2.3 Design 62](#_Toc489635176)

[16.2.4 Breadboard Layout 63](#_Toc489635177)

[16.2.5 Proto Layout 63](#_Toc489635178)

[16.2.6 Component List 63](#_Toc489635179)

[16.3 Software 63](#_Toc489635180)

[16.3.1 Requirements 63](#_Toc489635181)

[16.3.2 Design 63](#_Toc489635182)

[16.3.3 Memory Usage 63](#_Toc489635183)

[16.3.4 Timing Performance 63](#_Toc489635184)

[16.4 Testing 63](#_Toc489635185)

[16.4.1 Unit Tests 63](#_Toc489635186)

[16.4.2 Integration Tests 63](#_Toc489635187)

[Appendix A: New device template 64](#_Toc489635188)

[17 New Device Template 64](#_Toc489635189)

[17.1 Generic 64](#_Toc489635190)

[17.1.1 Requirements 64](#_Toc489635191)

[17.2 Hardware 64](#_Toc489635192)

[17.2.1 Requirements 64](#_Toc489635193)

[17.2.2 Inputs/Outputs 64](#_Toc489635194)

[17.2.3 Design 64](#_Toc489635195)

[17.2.4 Breadboard Layout 64](#_Toc489635196)

[17.2.5 Proto Layout 64](#_Toc489635197)

[17.2.6 Component List 64](#_Toc489635198)

[17.3 Software 64](#_Toc489635199)

[17.3.1 Requirements 64](#_Toc489635200)

[17.3.2 Design 64](#_Toc489635201)

[17.3.3 Memory Usage 64](#_Toc489635202)

[17.3.4 Timing Performance 65](#_Toc489635203)

[17.4 Testing 65](#_Toc489635204)

[17.4.1 Unit Tests 65](#_Toc489635205)

[17.4.2 Integration Tests 65](#_Toc489635206)

# List of Tables

[Table 1: Open Issues 2](#_Toc489635207)

[Table 2: History 2](#_Toc489635208)

[Table 3: Abbreviations 2](#_Toc489635209)

[Table 4: Glossary 2](#_Toc489635210)

[Table 5: Useful Links 3](#_Toc489635211)

[Table 6: Devices 15](#_Toc489635212)

[Table 7: Device Descriptions 16](#_Toc489635213)

[Table 8: Chapter Division 16](#_Toc489635214)

[Table 9: Requirements G 18](#_Toc489635215)

[Table 10: Requirements GH 18](#_Toc489635216)

[Table 11: Power Comparison 19](#_Toc489635217)

[Table 12: Power source location/type 20](#_Toc489635218)

[Table 13: Communication Comparison 20](#_Toc489635219)

[Table 26: Generic Diagnostics LEDs 25](#_Toc489635220)

[Table 14: Requirements GS 25](#_Toc489635221)

[Table 15: Requirements AP 29](#_Toc489635222)

[Table 16: Comparison Programming Languages 29](#_Toc489635223)

[Table 17: Requirements CH 30](#_Toc489635224)

[Table 18: Comparison External Memory 31](#_Toc489635225)

[Table 19: Comparison Communication Ext Application/PC 31](#_Toc489635226)

[Table 26: Controller Diagnostics LEDs 32](#_Toc489635227)

[Table 21: Diagnostics Controller 32](#_Toc489635228)

[Table 22: Components Controller 33](#_Toc489635229)

[Table 23: Requirements CS 33](#_Toc489635230)

[Table 26: Audio Diagnostics LEDs 36](#_Toc489635231)

[Table 26: DMX Diagnostics LEDs 38](#_Toc489635232)

[Table 26: Drums Trigger Diagnostics LEDs 40](#_Toc489635233)

[Table 26: Drums Trigger Diagnostics LEDs 42](#_Toc489635234)

[Table 26: GUI Diagnostics LEDs 44](#_Toc489635235)

[Table 26: Microphones Diagnostics LEDs 46](#_Toc489635236)

[Table 24: Requirements MG 48](#_Toc489635237)

[Table 25: Requirements MH 48](#_Toc489635238)

[Table 26: MIDI Diagnostics LEDs 49](#_Toc489635239)

[Table 27: Requirements MS 50](#_Toc489635240)

[Table 26: Audio Diagnostics LEDs 54](#_Toc489635241)

[Table 26: Pedals/switches Diagnostics LEDs 56](#_Toc489635242)

[Table 26: Remote Diagnostics LEDs 58](#_Toc489635243)

[Table 26:USB MIDI Diagnostics LEDs 60](#_Toc489635244)

[Table 28: : Requirements EG 62](#_Toc489635245)

[Table 26: Debug Diagnostics LEDs 62](#_Toc489635246)

[Table 26: NEW\_DEVICE Diagnostics LEDs 64](#_Toc489635247)

# List of Figures

# Introduction

Mestra stands for MESsage TRAnsformer.

It contains of several parts:

* External computer application, which is used to configure rules.
* Controller, which contains the rules and receives, processes and sends messages to the slave devices. This device is mandatory.
* Slave devices, which can be used to add outside interfaces (like MIDI, DMX etc) to the system. At least one slave device is needed to have a useful Mestra system.
* GUI, which is a special (slave) device to be able to see useful data displayed and enter (simple) rules.

## System Boundaries

Only Arduino(s) will be used. Rationale:

* Cost: This project is about learning about micro controllers/electronics. The Arduino is reasonable cheap compared to other micro controllers.
* Documentation: There is a lot of documentation about Arduino.
* Help community: There is a large forum community about Arduinos. For other micro controllers, the amount of help is less available.
* Electrically ‘forgiving’: The Arduino is known for not easy to ruin.
* Limiting for specs: The Arduino has very little of everything, this makes it challenging to find the boundaries and experience the problems using micro controllers with limited features. When needed, extra components or more feature-rich Arduinos can be used.

## One or more Arduinos

The initial idea was to use one Arduino. However, after several proof-of-concepts, it seems that using a single Arduino for the entire project is asking too much. Thus, the functionality will be split in multiple Arduinos. The advantage is, that each device can be built separately and later improvements can be made on a specific Arduino (prototype) instead of the entire system.

## Devices

In the table below, all devices (split in hardware/software) of the Mestra system are shown.

Table : Devices

|  |  |  |
| --- | --- | --- |
| **Abbr** | **Name** | **Description** |
| Con | Controller | Controller which controls all other Mestra devices. |
| Aud | Audio | Slave Mestra device for Audio input/output redirection. |
| Dmx | DMX | Slave Mestra device for DMX input/output. |
| Dpd | Drum Pad | Slave Mestra device for drum pads (in one box) |
| Dtr | Drums Trigger | Slave Mestra device for (acoustic) drums (or similar) triggering. |
| Gui | GUI | Mestra device for the GUI interface. |
| Mic | Microphones | Slave Mestra device for microphones input. |
| Mid | MIDI | Slave Mestra device for MIDI input/output. |
| Pds | Pedals/switches | Slave Mestra device for pedals and switches input. |
| Prx | Proximity | Slave Mestra device for proximity movements. |
| Rem | Remote | Remote control (device) for sending messages to the Controller. |
| Umd | USB MIDI | Slave Mestra device for USB MIDI input/output. |
| Dbg | Debug | Debug Mestra device for debugging purposes. |

As ‘proof of concept’, only the Controller (C), MIDI (M) and Debug (E) devices will be made.

Except for devices, also an external application is needed to configure all rules (triggers and commands).

### Device Descriptions

This paragraph shows per device a larger description.

Table : Device Descriptions

|  |  |
| --- | --- |
| **Device** | **Description** |
| Controller | This is a mandatory device; it receives from all other devices (slaves) messages, process them (depending on the configuration) and sends them to slaves. |
| Audio | This slave device is used for redirecting input audio to output audio, possibly mixing multiple input/output streams together. Input/output connectors: jack plugs / XLR. |
| DMX | This slave device is used for receiving and sending DMX signals, used for light systems. |
| Drum Pads | This slave device is a box containing a set of drum pads in one box. |
| Drums Trigger | This slave device is used for drum (or similar) triggers to send signals (e.g. to control MIDI drum sounds). |
| GUI | This is a special device that contains an LCD display, buttons and a cursor and/or a IR remote to manually program |
| Mic | Slave Mestra device for receiving inputs from (frequency band) microphones inputs. |
| MIDI | Slave Mestra device for MIDI input/output. |
| Pedals/switches | Slave Mestra device for pedals and switches input. |
| Proximity | Slave Mestra device for receiving inputs from proximity sensors. |
| Remote | Remote control (device) for sending messages to the Controller. |
| USB MIDI | Slave Mestra device for USB MIDI input/output. |
| Debug | Debug Mestra device for debugging purposes. |

## Chapter Division

The following abbreviations and chapters are used.

The abbreviations are used within the requirements.

Table : Chapter Division

|  |  |  |
| --- | --- | --- |
| Chapter | Abbr | Name |
|  | Gen | Generic requirements |
|  | GenH | Generic hardware requirements |
|  | GenS | Generic software requirements |
|  | App | (External Computer) Application requirements (only software) |
|  | ConG | Controller generic requirements |
|  | ConH | Controller hardware requirements |
|  | ConS | Controller software requirements |
|  | AudG | Audio generic requirements |
|  | AudH | Audio hardware requirements |
|  | AudS | Audio software requirements |
|  | DmxG | DMX generic requirements |
|  | DmxH | DMX hardware requirements |
|  | DmxS | DMX software requirements |
|  | DpdG | Drum Pads generic requirements |
|  | DpdH | Drum Pads hardware requirements |
|  | DpdS | Drum Pads software requirements |
|  | DtrH | Drums Trigger hardware requirements |
|  | DtrS | Drums Trigger software requirements |
|  | DtrG | Drums Trigger generic requirements |
|  | DmxH | DMX hardware requirements |
|  | GuiG | GUI generic requirements |
|  | GuiH | GUI hardware requirements |
|  | GuiS | GUI software requirements |
|  | MicG | Microphones generic requirements |
|  | MicH | Microphones hardware requirements |
|  | MicS | Microphones software requirements |
|  | MidG | MIDI generic requirements |
|  | MidH | MIDI hardware requirements |
|  | MidS | MIDI software requirements |
|  | PdsG | Pedals/switches generic requirements |
|  | PdsH | Pedals/switches hardware requirements |
|  | PdsS | Pedals/switches software requirements |
|  | PrxG | Proximity generic requirements |
|  | PrxH | Proximity hardware requirements |
|  | PrxS | Proximity software requirements |
|  | RemG | Remote generic requirements |
|  | RemH | Remote hardware requirements |
|  | RemS | Remote software requirements |
|  | UmiG | USB MIDI generic requirements |
|  | UmiH | USB MIDI hardware requirements |
|  | UmiS | USB MIDI software requirements |
|  | DbgG | Debug generic requirements |
|  | DbgH | Debug hardware requirements |
|  | DbgS | Debug software requirements |

# Generic

## AP, Hardware and Software

### Requirements

This paragraph shows all requirements for the entire project, both AP and controller/slaves.

Table : Requirements G

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Base ID** | **Version** | **Category** | **Item** | **Description** |
| Gen1 | - | 1.0 | Performance | Latency | Latency between a received message and sent (processed) messages related to the received message should not be more than 15 ms.  *Rationale: Studies are somewhat unclear, but generally a 15 ms latency is at the limit of what is musically noticeable.* |
| Gen2 | - | 1.0 | Performance | Startup | Startup of the system (in a typical setting) should take not more than 10 seconds.  *Rationale: Users do not expect startup times of ‘small’ devices to take long. In case of problems, resetting should be fast.* |
| Gen10 | - | 1.0 | Diagnostics | Error | Whenever an error occurs, the user will have enough information to find the root cause.  *Rationale: Error feedback is needed to solve a problem.* |
| Gen20 | - | 1.0 | Communication | - | The communication between all devices will be similar (both hardware/software). |

### Design

No generic design decisions (not handled by requirements already).

## Generic Hardware

### Requirements

Table : Requirements GH

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Base ID** | | **Version** | **Category** | **Item** | **Description** |
| GenH1 | - | | 1.0 | Power | Source | All devices should be able to be powered by a non USB power source.  *Rationale: USB power sources are mostly unavailable in a typical setup.* |
| GenH2 | - | | 1.0 | Power | Voltage | All components in the devices use a maximum of 5V.  *Rationale: Internally, 5 V->3.3 V adapters are used if needed.* |
| GenH3 | - | | 1.0 | Power | Current | 1A is the maximum current usage per device.  *Rationale: A default Arduino board powered by USB uses 0.5 A, most adapters are 1.0 A.* |
| GenH4 | - | | 1.0 | Power | Amount | The amount of power sources should be preferably 1 for all devices. *Rationale: in a typical setup, the least adapters, the best.*  **NOT MET** |
| GenH20 | Gen20 | | 1.0 | Communication | Hardware | The hardware communication will be equal for all devices (like either cabled or wireless like RF). |
| GenH10 | | - | 1.0 | Cost | Low | The cost of a single device should not exceed 25 euro (of component costs). |
| GenH20 | | - | 1.0 | Enclosure | Size | The enclosure size should not exceed 15 cm x 10 cm x 5 cm for all devices.  *Rationale: the devices will be placed on keyboards or put within limited space. Also, multiple devices will be used.* |
| GenH30 | | - | 1.0 | Safety | Damage | Connected devices should not damage the Mestra device.  *Rationale: Different devices can be connected.* |

### Design

#### Power

Regarding power, three options are taken into account to meet REQs GenHx. Note that due to REQ GenH1, USB power is not acceptable.

Note that MIDI Input power in the next table would only be supported by the MIDI device.

Table : Power Comparison

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item / Source:** | **220->12V Adapter** | **9V Battery** | **(MIDI Input power)** | **USB** |
| Minimum power | +++ | ++ | - | + |
| Cost of batteries | +++ | ---- | +++ | +++ |
| Continuity during usage | +++ | -- | - | +++ |
| Extra cable/plug needed | --- | + | +++ | -- |
| Wall outlet needed | --- | +++ | +++ | +++ |
| Reliable power | +++ | + | - | +++ |
| Enclosure modification | ++ | --- | --- | +++ |

To create a different power source for the MIDI device, is not selected, to keep all powering of devices consistent.

USB power is the default way of providing power while breadboarding. Depending on the slave type, USB power might be useful (e.g. for the MIDI slave, since most MIDI keyboards have MIDI). This is more convenient than having an adapter cable for example. In other cases USB power adapters (through wall sockets) can be used.

For other slaves, having a USB input for power is less needed, except for the fact that one wall outlet for multiple USB powered devices can be used (each giving 500 mA power).

A 9V battery has the possibility of being empty during use, which is totally unacceptable. Of course inserting a 9V battery before each use is a solution, but is quite costly.

On the other hand, the need of an extra socket and cable per device is not comfortable. Having a wireless communication between all devices would reduce this inconvenience.

Also, the power source depends on the location for the Controller or slave. See the following table.

Table : Power source location/type

|  |  |  |
| --- | --- | --- |
| **Device** | **Location** | **USB in neighbourhood** |
| Controller | Not relevant | No |
| MIDI | Near MIDI In/Out | Yes |
| USB MIDI | Near USB In/Out | Yes |
| DMX | Near DMX controller/light | No |
| Drum Pads | Near drums | No |
| Drums Trigger | Near drums | No |
| Pedal/Switches | Near pedals/switches on floor | No |
| Remote | Can be anywhere | No |
| Audio | Near guitar gear/floor | No |
| Microphones | Can be anywhere | No |
| Proximity | Can be anywhere, high likely near keyboard | No/Yes |
| GUI | On keyboard / in sight | Yes |
| Debug | - | Yes |

The Remote should be very mobile (to be used by singers and other musicians), preferably a battery powered device would be most convenient. This also is the case for the Drums trigger.

**Conclusion:** A 12V, 1A adapter and/or USB will be selected for all devices. Optionally, a special adapter with split outputs can be used to power multiple devices (in close range) with only one adapter.

12V adapters can result in some heat, since 7V for the input pin of an Arduino is requested to get a 5V output. This results in some unused heat though.

#### Communication from Controller to slaves

##### Comparison

Many options are available for the communication between the controller and slaves. The most important is REQ Gen1.

Table : Communication Comparison

|  |  |  |  |
| --- | --- | --- | --- |
| **Item \ Type** | **Cables** | **WIFI** | **RF** |
| Cost | - | ++ | +++ |
| Setup Time | --- | +++ | +++ |
| Speed | +++ | + | ?? |
| Reliability | ++ | - | + |
| Software | + | -- | -- |
| Extra connections | --- | +++ | +++ |
| Soldering work | --- | ++ | ++ |

Cables have as highest advantage, that these are reliable. However, all devices need to communicate with the Controller, so it will mean one cable per slave. And either two connections per slave to make some kind of network of many connections on the Controller. When multiple slaves of the same type should be connected, a network is the only possibility.

TODO: additional checks are needed:

* For cables:
  + What kind of protocol: SPI (already used by SRAM/SD) or I2C?
  + What connectors (Dsub?)
* For WIFI:
  + Needs for a router or can a local network be created?
  + Is speed high enough?
  + ESP8266 need to be programmed probably
* RF:
  + Check in live situations with lots of noise/other RF sources
  + Is speed high enough?
  + Free channel checking need to be implemented

##### Network Topology

The following requirements are needed for the network:

* The Controller sends messages to each slave device.
* Each slave device sends messages to the Controller.
* Slave devices do not send messages to other slave devices.

##### Slave Presence

Initially, the Controller needs to know which slave devices are present. Therefore, each device has its own predefined channel. TODO: how to define this / (auto) changing of channels in case of wireless communication.

During the initialization the following actions take place:

* The Controller polls for used devices/channels
* Each device returns its type (e.g. MIDI, Pedals/Switches etc)
* The Controller sends information to each device for what messages to send data back. This depends per slave.

##### Loop

After initialization, again the Controller always takes initiative. The reason is that important messages (like from MIDI) should be handled before lower priority messages (like a message from a pedal/switch) and important messages should not be interfered by possibly colliding messages from less priority slave devices.

Therefore the Controller polls each device for messages to be received. Maybe high priority devices will be polled more often than less priority devices.

Assuming all devices have same priority, the following flow will be typical for two slave devices:

1. The controller asks slave 1 if there are messages (Controller is transmitter)
2. The slave sends messages back (including a last (empty) message) (Slave 1 is transmitter)
3. Assuming these are high priority messages, messages are sent back. It can happen that some messages are discarded (e.g. MIDI CC messages with the same CC value).  
   It should be prevented that switching the transmitter happens too often).
4. Next slave is being handled (note that high priority slaves might be polled more often).

##### RF Proof of Concept

There are different types of RF, like below 1 GHz and above 1 GHz frequencies. Normally, 2.4 GHz is used. The default module to be used is NRF24L01+, however also the cheaper SE8R01 is available. Both will be used for the proof of concept.

* Which RF
* Used communication method: SPI
* Speed: Highest reliable / highest possible (cost more energy, but since an adapter will be used, this is not a problem). Default speeds are 250 mbps, 1 gbps, 2 gbps.
* Switching between receiving and transmitting.

##### NRF24L01 / BE8… TODO

These are the typical device used for 2.4 GHz wireless communication. Since the latter is cheaper and compatible, the latter is chosen. Also, it has a special network topology, using a 6-1 network for 6 transmitters and 1 receiver. The idea is to use this to:

* Let each slave be a transmitter
* The controller is the (only) receiver
* Each slave sends messages as soon as an input signal is received that has to be sent to the controller.
* After each message sent by a slave, the slave

##### One or Two RFs per device

After a short check it turned out that sending a message can take from 0,6 ms (16 byte payload) to several ms when retries are needed. Without retries sometimes messages are lost. However, several ms is too much to delay a message. Therefor an own protocol will be used.

Also to prevent switching (which takes 250 to 350 us excluding overhead) it is best to use 2 RF’s per device, one for sending. This means that devices can both transmit and receive without switching.

Also this makes it easier to send whenever needed, in case of collisions do a (smart) resend.

##### Best Channel / Initialization

The 2.4 GHz band used by the nRF24L01+ radios is from 2.4 GHz and has 125 possible channels. From these channels until channel with frequency 2.484 GHz is used by WIFI and therefore unusable by Mestra (also because WIFI will be heavily used by changing audience and thus very unpredictable).

The controller will check which frequency is best and send it to all slaves which can send a message back when received to move to the new frequency.

TODO: Check if the used frequency needs to be changed during operation.

This way the controller automatically knows which devices are present (the slave devices return their type).

##### Normal Operation

Devices will send messages when available. This means it can happen that messages collide. If so, there is a default scheme how long each device waits until trying to retransmit (varying from 10 us to 100 us).

The time to send a 8 byte payload message cost (see datasheet, page 38, fragment below).

Since ACKs will not be used, the time will be T\_UL + 2 \* T\_stdby2a + T\_IRQ = PL / SPI\_data\_rate + 2 \* 0,130 us + 0,006 us =113 / 16 us + 2 \* 0,136 us = around 7 us. However when tested, the shortest time was around 136 us (so let’s assume that).

Where PL (payload length) = 8 \*( 1 + 3 (address) + 8 + 1 (CRC) )+ 9 (bits) = 113 bits.

The SPI rate is assumed 16Mbps

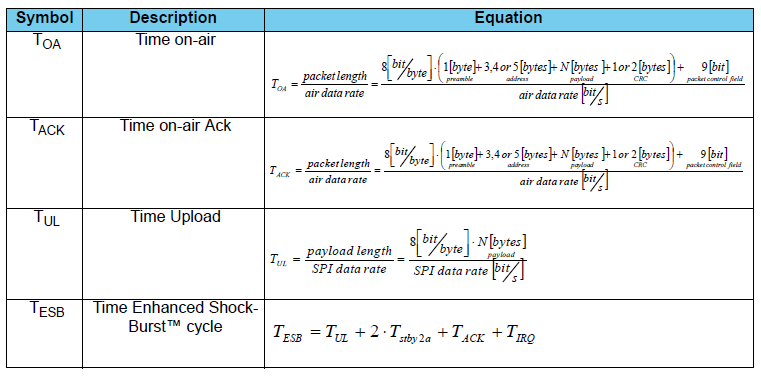


Figure : Time Equations

* This means per second 1000000 / 136 / 2 = 3676 messages can be sent. The division by 2 is because for each message an acknowledgement message is sent. Retries are not taken into account here.

##### Acknowledgements

Every message has to be acknowledged, both from controller and from slaves. An acknowledgement command has two bytes: one Ack ID and the sequence number to be acknowledged. The other 6 bytes can be used for normal messages which otherwise had to be sent normally.

Acknowledgements need to be sent less than 0.5 ms after receiving the message, to prevent the sender sending an equal message, thinking it has not arrived.

##### Reducing Messages

To reduce messages multiple commands can be stored in one payload.

Also messages to be sent are not sent within a 2 ms from one device. Messages are ‘saved’ until ready to be sent after 2 ms.

##### Multiple Messages

All commands either have a fixed length or their length inside the command as first byte(s) after the command ID. This means the receiving device knows if a message is unfinished and has to wait for more. In this case an ACK is sent only after the last one (last message). This prevents collisions and reduces messages.

##### Retries

It can happen that retries are need. For example, when the receiving device does not get a message, or when an acknowledgement message is not received back. 2 ms after sending a message and not getting an acknowledgement message, the message is sent again. An acknowledgement message does not need to be acknowledged, since the sender of the original message will send its message again after 2 ms, and continues to do so until the message acknowledge has been received.

Note that it takes about 0.5 ms to send a message, the acknowledge message will be sent by the receiver after a very short time (< 0.5 ms) and sending takes another 0.5 ms. So 2 ms should be adequate to be sure the message has not been received.

##### Payload

This depends on the type of device. All messages are 8 bytes. In case bigger messages need to be sent, they are split up (e.g. MIDI system exclusive messages). The partial messages need to be stored until complete.

Device IDs do not need to be sent separately, but a message ID is needed for the acknowledgement. Therefore each first byte of the payload is the sequence number for that device message counter.

The other bytes are e.g. for a MIDI device, Note on command: 2nd byte: NoteOn, 3th byte: Note number, 4th byte: velocity.

Multiple MIDI messages can be packed in one payload.

##### Alive Message

If the controller has not been sending a message to a slave for 5 seconds, it will send an alive message. If there is no reaction, both the controller and slave will notify it with a LED.

#### Diagnostics LEDs

All devices will have a LED to show the power. The reason is that almost all boxes will not have any direct notification if it has power or not.

Also, each box communicates through RF, so for this reason a LED will show the RF status. Note that when the controller polls each device, this will not be shown (since it happens every few ms), and slave devices receiving a poll request for packages and no messages are available (thus an ‘empty’ ACK package is sent, will also not result in a LED being on at the slave side, since this will happen many times per second.

Table : Generic 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 other device(s).  Triple fast blinking per second: problem with RF  On: non empty message transmitting/receiving |

Note that if the GUI Device shows errors whenever possible.

Blinking details:

* Slow blinking means one blink of 20 ms, followed by a pause of 980 ms (totaling 1 s).
* Double fast blinking means two blinks of 20 ms with a 80 ms gap, followed by a pause of 800 ms (totaling 1 s).
* Triple fast blinking means three blinks of 20 ms with a 80 ms gap, followed by a pause of 700 ms (totaling 1 s).

## Generic Software

### Requirements

This paragraph shows all requirements common to all Mestra (embedded) software.

Table : Requirements GS

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Base ID** | **Version** | **Category** | **Item** | **Description** |
| GenS100 | - | 1.0 | Rules | Amount | At least 100 rules can be defined.  *Rationale: to have a useful system, enough programming flexibility is needed.* |
| GenS200 | - | 1.0 | Rules | Simultaneously | At least 20 rules can be active simultaneously.  *Rationale: to have a useful system, enough programming flexibility is needed.* |
| GenS210 | - | 1.0 | Triggers | Amount | At least 100 triggers can be defined.  *Rationale: to have a useful system, enough programming flexibility is needed.* |
| GenS220 | - | 1.0 | Commands | Amount | At least 100 commands can be defined.  *Rationale: to have a useful system, enough programming flexibility is needed.* |
| GenS230 | - | 1.0 | Commands | Per Trigger | At least 10 commands per trigger can be defined.  *Rationale: to have a useful system, enough programming flexibility is needed.* |
| GenS300 | - | 1.0 | Communication | No noise from other devices | Slaves should work also when other (non Mestra) devices using the same type of communication are present.  *Rationale: when using wireless communication, other RF or WIFI should not interfere with Mestra devices.* |
| GenS310 | - | 1.0 | Communication | No noise to other devices | Other devices should not be interfered more than needed by the use of Mestra devices.  *Rationale: except for the channels/ports/resources used by Mestra devices, all other non Mestra devices using the same communication should not be interfered with.* |
| GenS320 | G20 | 1.0 | Communication | Protocol | The communication protocol for all devices will be similar.  *Rationale: Adding devices need to communicate with the controller.* |
| GenS400 | G1 | 1.0 | Performance | Prevent Messages | Prevent sending messages from a slave to the Controller when not needed.  *Rationale: Sending (and the resulting received message(s)) cost a lot of time, also it pollutes the bandwidth.* |
| GenS500 |  | 1.0 | Testing | Unit testing | Testing of software can be performed automatically by using unit testing.  *Rationale: Manual testing is too time consuming and software will be complex.* |

### Design

#### Slave Loop

Each slave has the same loop:

* If slave has an incoming message (from the controller):
  + Read incoming messages
  + Convert to specific signal/message for device
  + Send signal/message to device specific output(s)
  + Show notification (LED)
* If device has a incoming (device specific) signal/message
  + Convert to generic message
  + Send to Controller
  + Show notification (LED)

#### Preventing unnecessary messages

For GenS400, unnecessary messages should not be sent. Unnecessary messages are messages, which will not be changed by the Controller, and it is known beforehand.

However, for this requirement, some intelligence need to be moved to the slaves:

* Each slave should know for which type of messages sending to the Controller is not needed.
* In this case, the device possibly has to ‘forward’/return the message to its output(s).

Also, as an additional refinement, the device can be forwarded/return its message (without any change), BUT still send it to the Controller (to be processed further). For e.g. MIDI Note On commands this will ensure a better latency (while still the Note On can result in additional messages).

The refinement results in:

* The Controller needs to send to each device a list of messages to handle itself, send to the controller, or both.
* Each device needs to store this list and act upon it.
* Note that the list can be changed realtime (through the Controller).

#### IDE

The default IDE to be used for Arduino is the Arduino IDE. However, this IDE has the problem that when using more than approximately 10 files, the file names do not fit in the upper status bar. Therefore, another alternative will be chosen.

There are many options, like Eclipse, Microsoft Visual Studio plugins, and plugins for editors. Since I am used to Visual Studio, I selected the VisualMicro extension for Visual Studio.

#### Folder structure

The following folder structure will be used:

* Mestra
  + Documents
    - Manual
    - Design document (this document), in the future split up by device.
    - Excel sheet for design document
    - (Future) Ideas
  + Code
    - Project file for Visual Studio
    - Production
      * Common
      * Controller
      * Midi
      * Debug
    - Test
      * Common
      * Controller
      * Midi

## Testing

### Unit Tests

Because unit testing is needed in an automated way, classes should not use Arduino.h (or Arduino.h has to be stubbed). All tests will be performed within Visual Studio in a separate Test Project, runnable without hardware.

### Integration Tests

All devices will have a set of (manual) integration tests. Mostly this will involve the Controller.

# Application (AP)

## Requirements

Table : Requirements AP

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Base ID** | **Version** | **Category** | **Item** | **Description** |
| App100 |  | 1.0 | User | Friendly | The application should be user friendly.  *Rationale: Users want an easy way to define rules and are not always technical experts within the MIDI domain.* |
| App110 |  | 1.0 | Platform | Independency | The application should be supported for various operating systems, at least Windows and Mac OS.  *Rationale: Many users into music use either Windows or Apple (Mac OS).* |

## Design

### Programming Language

For Since the user interface of the external application will be GUI-heavy, a high level (object oriented) language is desired.

For REQ App110, a programming language needs to be used that support multiple platform. C# / C++ with XAML/WinForms are not good candidates, since Macintosh is not supported.

Java is a candidate. Possible GUIs are:

* TODO

Python is a candidate. Possible GUIs are:

* TODO

Below advantages between languages/GUIs can be checked.

TODO

Table : Comparison Programming Languages

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Java / |  |  | Python / TkInter |
| Experience |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

## Testing

### Unit Tests

TODO

### Integration Tests

TODO

# Controller

## Generic

### Requirements

None.

## Hardware

### Requirements

Table : Requirements CH

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Base ID** | **Version** | **Category** | **Item** | **Description** |
| ConH1 | Gen1, GenS1,  GenS10, GenS20, | 1.0 | External Memory | Amount | The memory size should be high enough to store all rules (triggers/commands).  *Rationale: To meet REQ Gen1, data cannot be loaded from a slow component (SD card).* |
| ConH2 | Gen2 | 1.0 | SD | Amount | The time to copy all rules to external memory should take less than 5 seconds.  *Rationale: another 5 seconds is left for software initialization and device (wireless?) coupling.* |
| ConH10 | Gen10 | 1.0 | Diagnostics | Error | Whenever an error occurs, LEDs will be used to show the root cause. |

### Inputs/Outputs

The controller will have the following inputs:

* USB, only used to flash the sketch.
* Adapter, for power.
* SD card, for the SD card containing the configuration file.
* Wifi (wireless), for wireless transmission of the sketch.
* RF (wireless), for receiving messages from slave devices.

The controller will have the following outputs:

* RF (wireless), for sending messages to slave devices.
* LEDs, for receiving/transmission and diagnostics information.

### Design

#### Arduino Type

Due to REQ ConH20, 2 KB for an Arduino Uno is on the low side, therefore an Arduino Mega is chosen.

Also, during development, it is comfortable to have the RX available for debugging, and having multiple UARTs is convenient.

Most important, the Arduino Mega has 256 KB flash instead of 16 KB. It is expected, the implementation of all commands, triggers and rules may be consuming way more than 16 KB.

#### External memory selection

Excel sheet <TODO> shows 128 KB is enough to store everything according REQ ConH1.

An Arduino Uno has 2 KB, an Arduino Mega 4 KB internal memory. Therefore, external memory is needed.

Two types of memory are taken into account:

Table : Comparison External Memory

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item** | **3K256 SRAM** | **23LC1024 SRAM** | **LP16040 (TODO)** | **EEPROM** |
| Reading speed | +++ | +++ | - | +++ |
| Writing speed | +++ | +++ | + | - |
| Wear-off | +++ | +++ | +++ | --- |
| Ease of use | -- | +++ | --- | + |

The reading speed of LP16040 is low, since it is a DIP24 IC, which needs either lots of digital pins, or a shift register, resulting in low read/write times.

EEPROM has problems when needing to write often that is wears off the IC.

3K256 SRAM has only 32 KB and 4 ICs are needed, with logic to combine them.

Therefore, it is clear a 23LC1024 is the best solution.

#### SD Card

Due to CH210, the SD card should be copied to external memory within seconds.

The typical SD card speed using SPI on an Arduino is like 100 KB/s, reading 128 KB results in approximately 1 second, which is well within spec.

#### Communication between external application and PC

The configuration file generated from the external application needs to be stored on the controller. This can be done in multiple ways:

Table : Comparison Communication Ext Application/PC

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **SD** | **USB cable** | **Wireless** |
| Ease of use | -- | + | +++ |
| Extra hardware needed | +++ | +++ | --- |
| Enclosure complexity | -- | +++ | + |
| Reliability | +++ | +++ | + |
| Usability | +++ | +++ | -- |
| Software complexity | +++ | --- | --- |

It shows a USB cable is the best option. However, making a SD opening in the enclosure of the controller, makes it possible to use both the SD and USB cable option.

The wireless option will be not implemented, since it might well be possible a wireless network is present (although very likely during creating the configuration file). But except the transferring of the configuration file, it has no further use. Also using an RX/TX signal, it might be problematic when using RX/TX for another wireless solution to/from the slaves (see next paragraph).

#### Diagnostics LEDs

Table : Controller Diagnostics LEDs

|  |  |  |
| --- | --- | --- |
| **Function** | **LED Color** | **Description** |
| Power | Blue (generic) | Off: Power off  On: Power on |
| RF, one per slave (max 5) | 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.

#### Diagnostics/Errors

Available LEDs:

* On/Off (blue)
* Receiving (green)
* Sending (red)

Whenever an error occurs, LEDs will display as shown below.

Table : Diagnostics Controller

|  |  |  |
| --- | --- | --- |
| **Root Cause** | **LEDs** | **Solutions** |
| SD Card cannot be read | On/Off: Blinking  Receiving: Off  Sending: Off | Reinsert the SD card and repower the device. |
| SRAM not accessible | On/Off: Blinking  Receiving: Off  Sending: On | None. |
| RF not accessible | On/Off: Blinking  Receiving: On  Sending: Off | None. |

Note that if the GUI Device shows errors whenever possible.

### Breadboard Layout

The breadboard will contain the RF communication breakout board, the adapter for it, and the SRAM chip including LEDs. This will take up maximum a half bread board.

### Proto Layout

TODO

### Component List

The cost of a prototype will be approximately (see Excel document, tab Cost):

Table : Components Controller

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item** | **Item Cost** | **Amount** | **Total** | **Datasheet** |
| Arduino Mega | € 7,35 | 1 | € 7,35 |  |
| SD Recorder Shield | € 1,50 | 1 | € 1,50 |  |
| SD Card 128 MB | € 3,00 | 1 | € 3,00 |  |
| Enclosure | € 5,00 | 1 | € 5,00 |  |
| SRAM | € 3,50 | 1 | € 3,50 |  |
| 2 x RF Transceiver | € 0,80 | 2 | € 1,60 |  |
| Various electronics | € 2,00 | 1 | € 2,00 |  |
| Enclosure | € 5,00 | 1 | € 5,00 |  |
| **Total** |  |  | **€ 28,95** |  |

## Software

### Requirements

Table : Requirements CS

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Base ID** | **Version** | **Category** | **Item** | **Description** |
| ConS1 | GenH10 | 1.0 | Flash | Amount | The amount of Flash memory is limited.  *Rationale: Cost perspective.* |
| ConS10 |  | 1.0 | SRAM | Usage | The total combination of triggers, rules and commands are only limited by the storage (there should be no predefined mapping).  *Rationale: More flexibility* |

### Design

The controller has by far the most extensive sketch. The reason for this is, that all intelligence is built into this device. It receives messages, transforms them, and sends them to the correct slave(s).

### Memory Usage

#### Introduction

A lot of information need to be stored. Because of REQ ConS10, the default 8 KB SRAM of the Arduino Mega is not enough.

The internal 8 KB SRAM will be used for only the highest necessary data (REQ ConS1)

* Stack trace (can be larger than other modules, due to the complexity of rules and the OO kind of programming)
* Buffers (initially 512 bytes for the SD card -> SRAM buffer, these buffers will be removed after copying).
* Local variables

Therefore, none other lists should be saved in SRAM. All lists will be stored in external SRAM (unless REQ Gen1 is affected).

The external 128 KB SRAM will be used for storing the rules and commands for all (slave) devices. To be most flexible, the SRAM will not be divided evenly per slave, but used as needed.

Also, memory consumption should be kept low.

In the following paragraphs, per slave device the memory layout/usage is explained.

#### Trigger filtering

The Controller will send messages to the slaves for which signals messages will be sent towards the Controller. This way the Controller will only get messages that really needs to be processed, reducing the number of packages to be sent by the RF network and keeping the processor of both Controller and slaves lower.

This means, that the Controller does not need to store the status of all slaves (like the note or CC values for the MIDI slave, or the current signals for switches and pedals). This means all trigger conditions will be stored by the slaves.

What needs to be stored are all trigger tables themselves, and all commands. Below per slave the memory usage/mapping will be explained.

#### Generic

TODO

#### MIDI

#### Pedals/switches

Assuming there will be 8 pedals and 8 switches, the size needed is 8 (switches) \* 2 (type on/off) \* 2 (address size) + 8 (pedals) \* 3 (types) \* 2 (address size) = 32 + 48 = 80 bytes, which is negligible.

### Timing Performance

TODO

## Testing

### Unit Tests

The Controller has the most complex software thus testing is critical.

The classes regarding rules, commands, triggers etc. are not using Arduino specific code. This means the tests can be created by running them on a standard PC.

### Integration Tests

The following tests will be performed (by connecting a keyboard/synthesizer):

1. Play a note, same note should be played.
2. Play many notes, all notes should be played, no notes should be heard after releasing all notes.
3. As 1, while moving the joystick.
4. As 2, while moving the joystick
5. As 1, while using aftertouch
6. As 2, while using aftertouch
7. Specific tests for rules, commands etc. TODO

RF Interference tests:

1. Testing with one or more mobile phones communicating via WIFI
2. Testing with one or more mobile phones communicating via Bluetooth
3. Testing while one or more mobile phones are starting to communicate via WIFI
4. Testing while one or more mobile phones are starting to communicate via Bluetooth
5. Putting a microwave near the controller or slave devices

# Audio

## Generic

### Requirements

TODO

## Hardware

### Requirements

TODO

### Inputs/Outputs

TODO

### Design

#### Diagnostics 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

TODO

### Timing Performance

TODO

## Testing

### Unit Tests

TODO

### Integration Tests

TODO

# DMX

## Generic

### Requirements

TODO

## Hardware

### Requirements

TODO

### Inputs/Outputs

TODO

### Design

#### Diagnostics LEDs

Table : DMX 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

# Drum Pad

## Generic

### Requirements

TODO

## Hardware

### Requirements

TODO

### Inputs/Outputs

TODO

### Design

#### Diagnostics LEDs

Table : Drum Pads 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

# Drums Trigger

## Generic

### Requirements

TODO

## Hardware

### Requirements

TODO

### Inputs/Outputs

TODO

### Design

#### Diagnostics LEDs

Table : Drums Trigger 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

# GUI

## Generic

### Requirements

TODO

## Hardware

### Requirements

TODO

### Inputs/Outputs

TODO

### Design

#### Diagnostics LEDs

Table : GUI 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

# Microphones

## Generic

### Requirements

TODO

## Hardware

### Requirements

TODO

### Inputs/Outputs

TODO

### Design

#### Diagnostics LEDs

Table : Microphones 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.

Microphone data will not result in much data. There will be a number of microphones (based on frequency / frequency range) and regions of volume or edge detection. Assuming there are 3 microphones, each microphone can send either a value change or edge detection up or down change command (3 types), resulting in 3 (microphones) \* 3 (types) \* 2 (address size) = 18 bytes (negligible).

### Timing Performance

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

## Testing

### Unit Tests

TODO

### Integration Tests

TODO

# 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

Appendix: Add

Glossary:

* Device: A (hardware) box that belongs to the Mestra system
* Controller (Device): A device that controls slaves
* Slave (Device): A device that has inputs and/or outputs and is controlled by a Controller
* Message: A message that is transported from the Controller to the Slave or vice versa
* Trigger: A condition for an incoming message in the Controller that can fire a command.
* Command: A list of instructions that is executed when a trigger is fired.
* Instruction: A single instruction that is used to describe what happens if a trigger is fired.

Enclosure size midi:

* Size STM32F407VET6: 85 x 72 mm
* On MIDI device:
  + MIDI Inputs: 4 (Kronos, Piano, Behringer , Keytar or Drums, or Drums via Behringer)
  + MIDI Outputs: 2 (Kronos, …)
  + MIDI Thru: 0
  + Total: 6
* On MIDI Extender
  + MIDI Inputs: 4 (total 8)
  + MIDI Outputs: 6
  + MIDI Thru: 8
  + (Total: 18)

**Triggers**

**NoteOn**

<TriggerKeyword> <TriggerName> <NoteOnId> <McSpec> <NoteSpec> <ProgramName> <status>

@ means: case insensitive

<TriggerKeyword> = @<Trigger>

<TriggerName> = <identifier>

<NoteOnId> = @”NoteOn”

<McSpec> = ( [@“mc=” | “” ] <Mc>)

<Mc> = ( “1” | “2” .. | “15” | “16” )

<NoteSpec> = ( [@“Notes=” | “”]| <NoteRange>)) | ( [@“Note=” | “”] <Note>)

<NoteRange> = [<Note>] “~” [<Note>] | @”All” where Note1 <= Note2, Note >= “C0”

(lowest 12 notes are reserved?), Note <” G10”,

“~” means all remaining notes on either side

<Note> = (“C” | “D” | “E” | “F” | “G” | “A” | “B” ) (“#” | “b” | “”) (“-1” | “0” .. “7” | “8”], check value 1<=note <=127, see note numbers: <http://newt.phys.unsw.edu.au/jw/notes.html>

(typical piano keyboards with 88 keys are from A0 (note number 21) to C8 (note number 108).

<ProgramName> = <identifier>

<identifier> = <alpha\_char> + <id\_char>\*

<alpha\_char> = (“A”| “B” | ..| “Y” | “Z” | “a” | “b” | .. | “y” | “z” )

<digit> = (“0” | “1” | .. | “8” | “9”

<id\_char> = <alpha\_char> |<digit> | “\_” )

<status> = (@“enabled” | @“disabled”)

Examples:

* trigger TransposeTrigger NoteOn MC=1 Note=C1~G#5 Transpose5 disabled;

**NoteOff**

Similar as NoteOn except:

<TriggerKeyword> <TriggerName> <NoteOffId> <McSpec> <NoteSpec> <ProgramName> <status>

<NoteOffId> = @”NoteOff”

**Cc Trigger**

trigger <name> Cc <McSpec> <CcSpec> <ProgramName> <status>

<BnkPrgCheck> = ( @”BnkPrg” | “” ) ( <Mc> “.” <Bank> “.” <Prg> )

<Bank> = <digit> 0..127

<Prg> = <digit>0..127

**Translation to MC**

Always one MC

If note range = ALL or ~: create trigger in ALL-notes trigger table

Else if note range: create trigger in OCTAVE\_notes trigger table and specific NOTE trigger table \*

Else if single note: create trigger in single NOTE trigger table

\* example Note range F4~G7 results in 7 single-note triggers for F4 to B4 + 2 octave triggers for octave 5 and 6 and 8 single note triggers for C7-G7, total: 7 + 2 + 8 + 17.

The trigger table is defined by a hash key depending on MC, Note. Note can be:

Single note triggers: 0-127

Octave note triggers: 128-137 (octave 1..10)

All note trigger: 255

**When a note is received**

3 hash keys are calculated:

* All notes trigger table (using also MC)
* Octave trigger table (using also MC, note (octave))
* Single note trigger table (using also MC)

For all three tables, all commands are executed (and checked if the trigger is enabled and condition is met since other triggers can end up in the same trigger table).

**Performance**

To loop through 100 triggers within 3 tables, every trigger taking 10 instructions to check, taking 50 clock cycles. These are 50,00 clock cycle. Assuming 168 MHz this will cost 5,000/168,000,000 = 0.029 ms, thus very less.

The execution of the programs (assume 5 on average high), cost 500 instructions of 4 clock cycli each, resulting in 10,000 clock cycli, which results in 0.059 ms (assuming 168 MHz), also vey less.

**Memory usage**

Assuming there are 256 hash keys / trigger tables, and per entry the following information is stored:

* Type (MIDI/PedSw): 3 bits
* Enabled 1 bit
* For e.g. note on: MC 4 bits
* For e.g. note on: Note 8 bits (for oct)
* Program index 16 bits

Total: 32 bits -> 4 bytes

Thus total storage:

* Table start offsets: 256 \* 2 bytes (start) = 512 bytes
* Tables itself: 1,000 (entries) \* 4 bytes = 4,000 bytes

**Send DMX Instructions**

**Code examples**

send DMX Channel 128 Value 10

Send DMX Channel 128 Var V1

Send DMX Channel 128 Prop NoteNumber // Eg from MIDI

send DMX Channel 128 Values 10 20 30 40 50 60 70 80

send DMX Scenes 0 1 2 3 5 7 Mult 16 Value 10

send DMX OffsetChannel 128 Scenes 0 1 2 3 5 7 Mult 16 Value 10

send DMX OffsetChannel 128 Scenes 0 1 2 3 5 7 Mult 16 Values 10 20 30 40 50 60 70 80

**Code instructions**

0: Single channel

“send” “DMX” <channels\_str> <channel> <values\_str> <value>

1: Multiple channels

“send” “DMX” <channels\_str> <channel> <values\_str> (<value>)+

2: Multiple scenes, single value

“send” “DMX” <offset\_channel\_str> <offset\_channel> <scenes\_str> <scene>+ [<mult\_str> (8 | 16)] <values\_str> <value>

3: Multiple scenes, multiple values

“send” “DMX” <offset\_channel\_str> <offset\_channel> <scenes\_str> <scene>+ [<mult\_str> (8 | 16)] <values\_str> <value>+

<channels\_str> = (“channels” | “ch”), case independent

<channel> = <integer 0..255>

<value\_str> = (“values” | “val” | “v”), case independent

<value> = <integer 0..255>

<scenes\_str> = (“scenes” | “sc”), case independent

<scene> = <integer 0..6>

<mult\_str> = (“mult” | “m”), case independent

<mult> = <”8” | “16”>

<offset\_channel> = (“offset\_channel” | “offset\_ch” | “offch” | “och” | “oc”), case independent

<offset\_channel = <integer 0..2040 or 4080>, multiple of <mult>

**Instruction opcodes**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Byte** | **Bits** | **Meaning** | **Values** | **Meaning** | **Description/remarks** |
| 0 | 7~4 | Instruction Type | ???? | Send DMX |  |
| 0 | 3~2 | Sub command | 00 | Single Channel |  |
|  |  |  | 01 | Multiple Channels |  |
|  |  |  | 10 | Multiple scenes, single value |  |
|  |  |  | 11 | Multiple scenes, multiple values |  |
| 0 | 1~0 |  |  |  | Depending on Sub command |

**00: Single Channel**

0: Single channel

“send” “DMX” <channels\_str> <channel> <values\_str> <value>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Byte** | **Bits** | **Meaning** | **Values** | **Meaning** | **Description/remarks** |
| 0 | 1~0 | Value Type | 00 | Value |  |
|  |  |  | 01 | Variable |  |
|  |  |  | 01 | Property |  |
| 1 |  | Channel | 0-255 | Channel |  |
| 2 |  | Value / Property / Variable | 0-255 | Depending on b0.1~0 |  |

**01: Multiple channels**

1: “send” “DMX” <channels\_str> <channel> <values\_str> (<value>)+

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Byte** | **Bits** | **Meaning** | **Values** | **Meaning** | **Description/remarks** |
| 0 | 0~1 | MSB Channel | 00-11 | MSB Channel) | 0, 256, 512 or 1024 |
|  |  |  | 1 | MSB Channel (255-511) |  |
| 1 |  | Channel | 0-255 | Channel |  |
| 2 |  | NrOfValues | 0-255 | Number of values |  |
| 3..NrOfValues |  | Values | 0-255 |  |  |

**10: Multiple channels, single value**

2: “send” “DMX” <offset\_channel\_str> <offset\_channel> <scenes\_str> <scene>+ [<mult\_str> (8 | 16)] <values\_str> <value>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Byte** | **Bits** | **Meaning** | **Values** | **Meaning** | **Description/remarks** |
| 0 | 1~0 | Value Type | 00 | Value |  |
|  |  |  | 01 | Variable |  |
|  |  |  | 01 | Property |  |
| 1 | 7 | Multiplication factor | 0 | 8 |  |
|  |  |  | 1 | 16 |  |
| 1 | 6 | Offset channel or scenes | 0 | Byte 1 are scenes |  |
|  |  |  | 1 | Byte 1 is offset channel |  |
| 1 | 5~0 | If bit 1.6=0: Scenes |  | 6 scenes | (first) 6 scenes |
|  |  | If bit 1.6=1: Offset Channel | 0-255 | Offset channel multiplied by multiplication factor | If mult factor = 8: 0-512  If mult factor = 16: 0-1024 |
| 2 |  | Scenes |  | (LSB) scenes | (last) 8 scenes |
| 3 |  | Value | 0-255 | Value |  |

**11: Multiple channels, multiple values**

3: “send” “DMX” <offset\_channel\_str> <offset\_channel> <scenes\_str> <scene>+ [<mult\_str> (8 | 16)] <values\_str> <value> +

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Byte** | **Bits** | **Meaning** | **Values** | **Meaning** | **Description/remarks** |
| 0 | 1 | Multiplication factor | 0 | 8 |  |
|  |  |  | 1 | 16 |  |
| 0 | 0 | Offset channel or scenes | 0 | Byte 1 are scenes |  |
|  |  |  | 1 | Byte 1 is offset channel |  |
| 1 |  | If 0.0=0: Scenes |  | 8 scenes | (first) 8 scenes |
|  |  | If 0.0=1: Offset Channel | 0-255 | Offset channel multiplied by multiplication factor | If mult factor = 8: 0-2048  If mult factor = 16: 0-4096 |
| 2 |  | Scenes |  | (LSB) scenes | (last) 8 scenes |
| 3 |  | Number of Values | 0-255 | Number of values |  |
| 4..x |  | Values | 0-255 | Values |  |

* + N

**Hardware (Simple)**

Components

* STM32F103C8T6, 512 KB Flash, 20 KB SRAM
* SD Card
* 2x NRF24L01+

Inputs/outputs

* USB (no MIDI)
* SD slot
* 2 x MIDI In
* 1 x MIDI Out
* DMX Out
  + 2 x green: MIDI In
  + 1 x red: MIDI Out
  + 1 x red: DMX Out
  + 1 x blue: power

**Hardware (Max)**

Components

* STM32F446 (?), 1 GB Flash, 192 KB SRAM
* SD card (reader)
* CAN or RS485
* 2x NRF24L01+

Inputs/Outputs

* CAN/RS485
* USB (for MIDI/updates)
* SD slot
* 4x MIDI In
* 2x MIDI Out
* 2x DMX Out
* LEDS:
  + 4 x green: MIDI In
  + 2 x red: MIDI Out
  + 2 x red: DMX Out
  + 1 x blue: power
  + 6 x yellow/red: RF/comm
  + 1 x yellow/red: CAN/RS485

**Setup**

Kronos MIDI OUT -> Mestra MIDI IN 1

Studiologic MIDI OUT -> Mestra MIDI IN 2

Behringer MIDI OUT -> Mestra MIDI IN 3

(KeyTar -> Mestra MIDI IN 4)

Mestra MIDI OUT 1 -> Kronos MIDI IN

Mestra DMX OUT 5 -> DMX Chain

OR

Kronos MIDI OUT -> Mestra MIDI IN 1

Studiologic MIDI OUT -> Behringer MIDI IN

Behringer MIDI OUT -> Mestra MIDI IN 2 (merged with Studiologic)

Mestra MIDI OUT 1 -> Kronos MIDI IN

Mestra DMX OUT 3 -> DMX Chain

Memory for DMX:

128 channels -> 128 bytes

Add to preferences (max DMX channels used)