**Mestra**

**Combined Controller**

Design

Michel Keijzers, © 2017

# History

Table 1: History

|  |  |
| --- | --- |
| **Date/period** | **Actions** |
| Nov 8, 2017 | Initial version |

# Table of Contents

Contents

[History 1](#_Toc498126522)

[Table of Contents 2](#_Toc498126523)

[List of Tables 2](#_Toc498126524)

[List of Figures 2](#_Toc498126525)

[1 Introduction 3](#_Toc498126526)

[2 Folder Structure 4](#_Toc498126527)

# List of Tables

[Table 1: History 1](#_Toc498126528)

# List of Figures

**No table of figures entries found.**

# Introduction

## General

This document shows the design of the combined Controller and MIDI/DMX slaves.

## Typical Usage

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

# Requirements

# Workflow

## Introduction

This chapter describes in short what happens between creating a configuration file until the execution of instructions. As example a MIDI message will be used.

## Configuration File

The user writes a configuration file. As an example the next file is used. It will transpose all MIDI notes on MIDI channel 1 from notes C4 and higher, 5 semitones up.

Trigger TransposeC4Plus5 MC 1 C4~ TransposePlus5

Program TransposePlus5

Set $Note #NoteNr

Add $Note 5

Set #NoteNr $Note

Send

The trigger defines what happens when an incoming message is received, in this case a Note from C4 or higher on MIDI channel. What happens is that the TranpsoePlus5 program will be executed.

This program copies the property NoteNr (the key that has been pressed) to a variable note. To this note 5 is added (i.e. the terts) and is set to the NoteNr property. The message is than sent, which is a NoteOn command with the transposed note and the same velocity as the incoming message.

## Translation to Mestra File

The configuration file is translated by a PC application to a Mestra file. It contains of a list of triggers and programs (which contains instructions). However, to increase the performance, the NoteOn trigger is split in triggers for each separate note. To decrease the amount of memory, some notes are grouped together (each 8 notes). For the C4 note (and above), this means the following triggers are defined:

* Note 60 (C4)
* Note 61 (C#4)
* Note 62 (D4)
* Note 63 (E4)
* Note group 9 (which contains notes 64 to 71)
* Note group 10 (which contains notes 72 to 79)
* Note group 11 to 16 (which contains notes 80 to 127).

Thus the trigger is split into 4 + 7 = 12 Mestra triggers.

Note, there is also a All-Note trigger list, in case no note range is given.

The triggers (in our case only one) are translated to 5 byte triggers. In the Triggers document all triggers are defined.

The programs are also translated; this is defined by the instructions which are defined in the Instructions document.

The programs are not much more than ‘pointers’ to the first instruction of that program.

## Trigger Tables

There are trigger tables for each note, each note group (and all notes), for each MIDI channel. Also for other MIDI triggers there will be lots of trigger tables. Since it would be too memory consuming, the triggers are stored in a hash table. As a result, multiple different trigger types can end up in the same hash table, so the properties of the trigger need to be stored as well (and checked).

The size of the hash table (number of hash keys) is typically 256.

The formula to translate a trigger to a hash key is.

Hash\_key = based on depending on MC, Note (group) or other MIDI message type

Note (group): The lowest notes values are used for the groups (0 means all), 1-16 for note groups 1-16, notes 17.. for notes.

Other MIDI message type: e.g. pitch bend, after touch do not have a note number and will be treated differently for calculating the hash key. See table below.

|  |  |  |
| --- | --- | --- |
| **MIDI Type (4 bits)** | **Byte 1 for hash key** | **Byte 2 for hash key** |
| Note On/Off | MIDI channel | Note number\* |
| Polyphonic Key Pressure (aftertouch) | MIDI channel | Note number\* |
| Control Change | MIDI channel | Control number |
| Program Change | MIDI channel | Patch number |
| Channel Pressure / aftertouch | MIDI channel |  |
| Pitch Bend Change | MIDI channel |  |
| Channel Mode Messages | Control numbers (120-127) |  |
| System Common/RT Messages/Sysex | LSB 4 bits |  |

So 4 bits (MIDI Type) + Byte 1 and optionally byte 2 are used to generate the hash key using a simple XOR.

## Incoming message, trigger checking

When a MIDI note is received, let’s say G5 on MIDI channel 1, the following trigger tables are checked:

* MIDI, Note On, MC 1, Note G4 (note 67)
* MIDI, Note On, MC 1, Note group 9)
* MIDI, Note On, MC 1, All Notes

All triggers in each trigger table is checked if it is a MIDI Note On command trigger, and if it has the correct MC and note (group). Note there can be multiple triggers of the same type (e.g. Multiple Note On, MC1, Note G4 commands). For all triggers of the same type, the associated program is ran (mostly this will be only one, but does not need to be necessarily).

## Program execution

All instructions of the program will be executed. The length can be dynamical and depends on the content of the current instruction. The end of a program always is the instruction END.

# Design Decisions

## Memory Usage

Assuming there are 256 hash keys / trigger tables, and each trigger is 5 bytes:

Thus total storage:

* Table start offsets: 200 \* 2 bytes (start) = 400 bytes
* Tables itself: 1,000 (entries) \* 5 bytes = 5,000 bytes

Total: 5,400 bytes.

Assuming there are 200 programs, which 5 instructions with an average of 10 bytes per instruction. This results in 200 \* 5 \* 5 = 5,000 bytes.

## 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 very less.