

## MUX Library

Generated by Doxygen 1.8.9.1

Fri Sep 18 2015 19:01:51



# Contents

<b>1</b>	<b>Main Page</b>	<b>1</b>
<b>2</b>	<b>LICENSE</b>	<b>5</b>
<b>3</b>	<b>Class Index</b>	<b>7</b>
3.1	Class List . . . . .	7
<b>4</b>	<b>Class Documentation</b>	<b>9</b>
4.1	digit Struct Reference . . . . .	9
4.2	MUX Class Reference . . . . .	9
4.2.1	Member Function Documentation . . . . .	9
4.2.1.1	addAnode . . . . .	9
4.2.1.2	addCathode . . . . .	10
4.2.1.3	addDigit . . . . .	10
4.2.1.4	digitalWrite . . . . .	10
4.2.1.5	display . . . . .	10
4.2.1.6	invertAnode . . . . .	10
4.2.1.7	invertCathode . . . . .	10
4.2.1.8	off . . . . .	10
4.2.1.9	on . . . . .	10
4.2.1.10	setDigit . . . . .	10
4.2.1.11	setRow . . . . .	11
4.2.1.12	setValue . . . . .	11
4.2.1.13	start . . . . .	11
4.2.1.14	stop . . . . .	11
	<b>Index</b>	<b>13</b>



# Chapter 1

## Main Page

Task Manager based LED Multiplex library for chipKIT™.

### How it works

The **MUX** library is a fully static class which ties into the chipKIT API's *Task Manager* system.

The *Task Manager* is a system which allows the periodic execution of a function with fine grained (millisecond) resolution and accuracy.

The **MUX** library hooks into this system to update the display data to a multiplex of LEDs in any arrangement you require at a frequency of 500Hz. That is, one column (anode) of data is rendered ever 2ms.

The library is *anode-based*, in that the pin entries that are for the sake of convenience called anodes in the library are scanned in order and the pins which are nominated as cathodes have the data for the current anode presented on them. That doesn't mean that you *have* to use the anodes as connections to actual anodes, they could be used to drive cathodes and the cathode pins could be used to drive anodes. This is just purely the convention used throughout this library and its documentation.

### Using the library

To use the library you must first attach a number of pins as either anodes or cathodes. You do this using the functions:

```
MUX::addAnode(pin_number);  
MUX::addCathode(pin_number);
```

Call the functions as many times as required. The order you call the functions define the order in which the anodes are scanned and the order the display data is presented on the cathodes.

By default both the cathodes and the anodes will be set HIGH when active (that is, they will *source* current). If you need either of them to *sink* current (that is, be set LOW when active) you can invert the functionality of the pins:

```
MUX::invertAnode();  
MUX::invertCathode();
```

Throughout this document (and the library) any references to anodes and cathodes as parameters to a function refer to the *logical* anode or cathode number in the order that you called the addAnode or addCathode functions. The pin referenced in the first call to addAnode is anode 0. The pin referenced in the second call to addAnode is anode 1, and so on.

The library supports up to 16 anodes and up to 16 cathodes.

Once you have the pins defined you can *start* the display with the function:

```
MUX::start();
```

At any time you can stop the display refreshing (and detach the system from the task manager) with:

```
MUX::stop();
```

To get some data into the display you can either do it an LED at a time with the functions:

```
MUX::on(anode, cathode);
MUX::off(anode, cathode);
```

If you want to set an entire anode's worth of data at once you can use:

```
MUX::setRow(anode, value);
```

## Numeric display

It is possible to bind individual LEDs together to form 7-segment digital displays. To do this you need to create the digits in the system. Calling the *addDigit()* function will tell the library which pin combinations (anode and cathode) refer to which of the seven (or eight if you use the decimal point as well) LEDs that make up the digit.

For instance, if you have a setup where a 7-segment display is wired up to have the 7 main segments A through G (in standard 7-segment layout) on the pin combinations (anode,cathode) [(2,2), (2,3), (2,4), (2,5), (2,6), (2,7), (2,8)] you would call the *addDigit* function like this:

```
MUX::addDigit(2, 2, 2, 3, 2, 4, 2, 5,
              2, 6, 2, 7, 2, 8);
```

If you wanted the decimal point as well you would just add the pin combination that refers to that LED as the 15th and 16th parameters.

Digits are added in order from left to right.

You can set the content of any one digit using the *setDigit()* function. This function takes the digit number (starting from 0 and progressing in the order you added them) and the value to display. The value is made up of two parts packed into an 8 bit value.

The upper half of the value is called the *mode*. This defines how the digit will interpret the lower half of the value.

There are currently four modes:

- 0. Normal digit display (0-F)
- 1. Decimal point digit display (0.-F.)
- 2. Special character
- 3. Special function

There are three special characters defined for mode 2. The value portion corresponds to:

- 0. Degree sign
- 1. Minus sign
- 2. Equals sign

There is only one special function at the moment for mode 3:

- 0. Clear the digit

Here are some hexadecimal examples and the results on a digit:

- 0x04: 4

- 0x0E: E
- 0x1B: b.
- 0x17: 7.
- 0x21: -
- 0x22: =
- 0x30: (all segments off)

There is also a helper function which will take a numeric value and format it across all the defined digits. This takes an integer value (negative or positive) and displays the data as a decimal value right justified on all the digits.

```
MUX::setValue(value);
```





## Chapter 2

# LICENSE

Copyright (c) 2015, Majenko Technologies All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Majenko Technologies nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.



## Chapter 3

# Class Index

### 3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">digit</a>	.....	9
<a href="#">MUX</a>	.....	9



## Chapter 4

# Class Documentation

### 4.1 digit Struct Reference

#### Public Attributes

- uint8\_t **anode**
- uint8\_t **cathode**

The documentation for this struct was generated from the following file:

- MUX.h

### 4.2 MUX Class Reference

#### Static Public Member Functions

- static void [addAnode](#) (uint8\_t pin)
- static void [addCathode](#) (uint8\_t pin)
- static void [on](#) (int anode, int cathode)
- static void [off](#) (int anode, int cathode)
- static void [digitalWrite](#) (int anode, int cathode, int val)
- static void [setRow](#) (int row, uint16\_t data)
- static void [invertAnode](#) (boolean v=true)
- static void [invertCathode](#) (boolean v=true)
- static void [display](#) (int id, void \*tpr)
- static void [start](#) ()
- static void [stop](#) ()
- static void [addDigit](#) (uint8\_t aa, uint8\_t ac, uint8\_t ba, uint8\_t bc, uint8\_t ca, uint8\_t cc, uint8\_t da, uint8\_t dc, uint8\_t ea, uint8\_t ec, uint8\_t fa, uint8\_t fc, uint8\_t ga, uint8\_t gc, uint8\_t dpa=255, uint8\_t dpc=255)
- static void [setDigit](#) (int [digit](#), int data)
- static void [setValue](#) (int value)

#### 4.2.1 Member Function Documentation

##### 4.2.1.1 void MUX::addAnode ( uint8\_t *pin* ) [static]

Add an anode pin to the matrix.

**4.2.1.2** `void MUX::addCathode ( uint8_t pin ) [static]`

Add a cathode pin to the matrix.

**4.2.1.3** `void MUX::addDigit ( uint8_t aa, uint8_t ac, uint8_t ba, uint8_t bc, uint8_t ca, uint8_t cc, uint8_t da, uint8_t dc, uint8_t ea, uint8_t ec, uint8_t fa, uint8_t fc, uint8_t ga, uint8_t gc, uint8_t dpa = 255, uint8_t dpc = 255 ) [static]`

Add a digit mapping to the matrix.

A digit mapping consists of 7 or 8 pairs of anode and cathode numbers. These refer to the combinations required to access the segments A through G (and optionally the decimal point) on a 7-segment display.

**4.2.1.4** `void MUX::digitalWrite ( int anode, int cathode, int val ) [static]`

Control an individual LED in the same way as an IO pin

**4.2.1.5** `void MUX::display ( int id, void * tptr ) [static]`

This is an internal function and should never be called by a user. It is what does the actual displaying of data on the display. It needs to be public so the Task Manager has access to it.

**4.2.1.6** `void MUX::invertAnode ( boolean v=true ) [static]`

Invert the anode's output.

By default all anode pins are *active high* so they go HIGH when an anode is selected. This will invert that functionality so idle anodes remain HIGH and the active anode is switched to LOW.

**4.2.1.7** `void MUX::invertCathode ( boolean v=true ) [static]`

Invert the cathode's output.

By default all cathode pins are *active high* so they go HIGH when a cathode has its bit set. This will invert that functionality so unset cathodes are set HIGH and the active cathodes are switched to LOW.

**4.2.1.8** `void MUX::off ( int anode, int cathode ) [static]`

Turn off an individual LED in the matrix

**4.2.1.9** `void MUX::on ( int anode, int cathode ) [static]`

Turn on an individual LED in the matrix

**4.2.1.10** `void MUX::setDigit ( int digit, int data ) [static]`

Set an individual digit to a specific value or character.

The upper nibble of the data (*mode*) defines what the lower nibble (*value*) does.

- When *mode* is 0 *value* is a hexadecimal digit (0-F).
- When *mode* is 1 *value* is a hexadecimal digit with the decimal point turned on (0.-F.)
- When *mode* is 2 *value* is a special character:

- 0 = Degree Sign
- 1 = Minus Sign
- 2 = Equals Sign
- When *mode* is 3 *value* is a special function:
  - 0 = Clear the digit

#### 4.2.1.11 void MUX::setRow ( int *row*, uint16\_t *data* ) [static]

Set an entire row of data in the internal buffer

This corresponds to the data for one anode's worth of display data.

#### 4.2.1.12 void MUX::setValue ( int *value* ) [static]

Set a numeric value (integer) on the pre-defined digits.

#### 4.2.1.13 void MUX::start ( ) [static]

Start displaying data on the matrix

#### 4.2.1.14 void MUX::stop ( ) [static]

Stop displaying data on the matrix

The documentation for this class was generated from the following files:

- MUX.h
- MUX.cpp





# Index

- addAnode
  - MUX, [9](#)
- addCathode
  - MUX, [9](#)
- addDigit
  - MUX, [10](#)
- digit, [9](#)
- digitalWrite
  - MUX, [10](#)
- display
  - MUX, [10](#)
- invertAnode
  - MUX, [10](#)
- invertCathode
  - MUX, [10](#)
- MUX, [9](#)
  - addAnode, [9](#)
  - addCathode, [9](#)
  - addDigit, [10](#)
  - digitalWrite, [10](#)
  - display, [10](#)
  - invertAnode, [10](#)
  - invertCathode, [10](#)
  - off, [10](#)
  - on, [10](#)
  - setDigit, [10](#)
  - setRow, [11](#)
  - setValue, [11](#)
  - start, [11](#)
  - stop, [11](#)
- off
  - MUX, [10](#)
- on
  - MUX, [10](#)
- setDigit
  - MUX, [10](#)
- setRow
  - MUX, [11](#)
- setValue
  - MUX, [11](#)
- start
  - MUX, [11](#)
- stop
  - MUX, [11](#)