STM32F4-Discovery, 320x240 VGA

V1.0

Generated by Doxygen 1.8.14

Contents

1 Module Index

1.1 Modules

Here is a list of all modules:

2 Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

3 Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

4 Module Documentation

4.1 defines

Macros

- #define BUFFER_LENGTH 100

    Input buffer length.
- #define BUFFER_RESET 12

    Error code: command buffer overflow.
- #define COLORS 16

    Amount of colors.
- #define ERRORS 1

    Error feedback 1=on/0=off.
- #define LOW_PRIORITY 0

    NVIC priority value.
- #define MAX_BUFFERS 120

    Amount of commando's that can be buffered.
- #define MAX_COLOR_LENGTH 15

    Maximum string length for command types and colors.
- #define MAX_FILL_LENGTH 4

    Maximum fill length.
- #define MAX_INT_LENGTH 6

    Maximum string length for an int.
- #define MAX_TEXT_LENGTH 64

    Maximum text length.
- #define TIM5_PERIOD 84000

    Time period for the wait timer.
- #define TIM5_PRESCALE 2000

    Prescaler for the wait timer.
- #define TIM5_REP 1

    Repetition counter for TIM5.
- #define TYPE_NOT_FOUND 11

    Error code: type not found.

4.1.1 Detailed Description

Group of global defines.

4.1.2 Macro Definition Documentation

4.1.2.1 TIM5_PERIOD

```
#define TIM5_PERIOD 84000
```

Time period for the wait timer.

TIM5 basefreq = 2∗APB1 (APB1=42MHz) => TIM_CLK=84MHz, 84Mhz/84000/1 = 1kHz, 84Mhz = 84 MHz /x/ 84000 => x = ms

5 Namespace Documentation

5.1 IO Namespace Reference

namespace IO

Functions

- void delete_IO ()

  (Global) Deletes the IO layer.
- void init_IO ()

  (Global) Initiate the IO layer.
- int read (char ∗buf)

  (Global) Read from UART.
- void stop_Read ()

  (Global) Stops the UART::read() function.
- void write (char ∗text_out)

  (Global) Writes back through UART.

5.1.1 Detailed Description

namespace IO

Acts as a mediator between the UART and the rest of the program. Gets commands from the UI and the LL and parses it to UART.

5.1.2 Function Documentation

5.1.2.1 delete_IO()

```
void delete_IO ( )
```

(Global) Deletes the IO layer.

Deletes the IO-layer. Calls the UART::delete_UART() function.

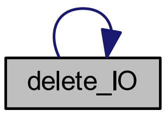
Parameters

void	

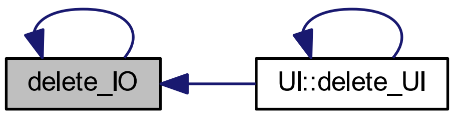
Returns

void

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.2 init_IO()

```
void init_IO ( )
```

(Global) Initiate the IO layer.

Calls the UART::init_UART function to start the UART. (Optional) call the UART::init_IDLE_Line() for inputs without the carriage return char.

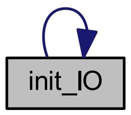
Parameters

void	

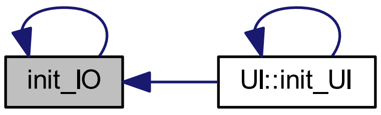
Returns

void

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.3 read()

```
int read (
             char * buf )
```

(Global) Read from UART.

Calls the UART::read() function of the UART. The user input is saved in the char ∗buf. The return state is whether a new user input was given(SET) or if the read was canceled (RESET).

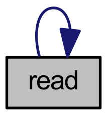
Parameters

char	∗buf Buffer to fill with the user input.

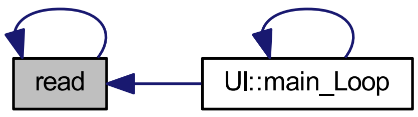
Returns

int exit_state returns whether the read functions was returned with new user input or by the stop_Read() function.

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.4 stop_Read()

```
void stop_Read ( )
```

(Global) Stops the UART::read() function.

Calls the UART::stop_Read() function to stop waiting for user input and check if there are buffered inputs.

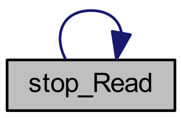
Parameters

void	

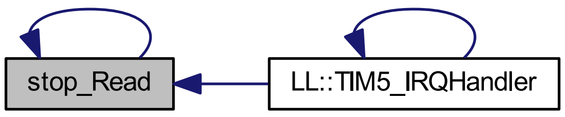
Returns

void

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.2.5 write()

```
void write (
            char * text_out )
```

(Global) Writes back through UART.

Write the char ∗text_out using the UART::write() function.

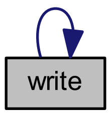
Parameters

char	∗text_out

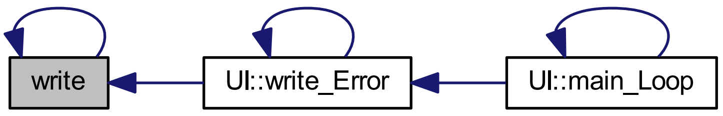
Returns

void

Here is the call graph for this function:



Here is the caller graph for this function:



5.2 LL Namespace Reference

Namespace LL.

Data Structures

- struct command_t

  Struct command_t for saving incoming commands.
- struct logic_t

  logic_t struct for the different flags and buffer.

Functions

- int color_To_Int (char ∗color)

  (LOCAL)Gets 8 bit rgb value from the colors name.
- void delete_LL (void)

  (GLOBAL)Destroy the LL
- int exec (void)

  (GLOBAL)executes the last command or the command buffer.
- void init_LL (void)

(GLOBAL)Initiate the LL.

- void init_TIM5 (void)

  (LOCAL)Initiate TIM5 for the wait_Ms() function.
- int set_Command (char ∗buf)

  (GLOBAL)Sets the command struct.
- void TIM5_IRQHandler (void)

  (GLOBAL) TIM5 interrupt handler for wait_Ms().
- void wait_Ms (int ms)

  (LOCAL)The wait function of the LL.

Variables

- const char ∗ colors [COLORS]

  List with all the color names.
- logic_t logic

  Struct with the LL variables.
- const int rgb [COLORS]

  List with all the color values.

5.2.1 Detailed Description

Namespace LL.

The logic layer get's the user input from the UI. The input buffer is split into sperate inputs and saved in a command_t struct. This is stored in the command buffer logic_t.buffers. after the command is set exec() is called if the logic level isn't waiting.

The function exec() sends the draw commands to the Vgascreen object. If there's more then 1 command buffered keep drawing until the buffer is empty, a wait is called or a repeat.

The Logic-layer uses TIM5 for the wait_Ms() function.

5.2.2 Function Documentation

5.2.2.1 color_To_Int()

```
int color_To_Int (
             char ∗ color )
```

(LOCAL)Gets 8 bit rgb value from the colors name.

Loops through the color names and checks whether a name is the same. Returns the 8 bit color value from rgb[].

The color names are:"zwart", "blauw", "lichtblauw", "groen", "lichtgroen", "cyaan", "lichtcyaan", "rood", "lichtrood", "magenta", "lichtmagenta", "bruin", "geel", "grijs", "wit".

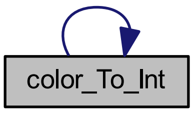
Parameters

char	∗colors

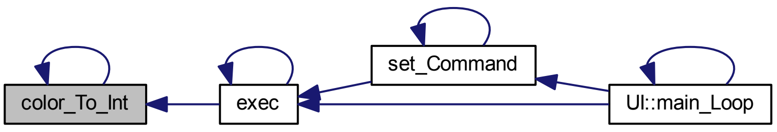
Returns

int color, the color value from 0-255.

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.2.2 delete_LL()

```
void delete_LL (
             void  )
```

(GLOBAL)Destroy the LL

Deletes and Vgascreen object and resets all the flags.

Gives the TYPE_NOT_FOUND error if a wrong input is given or the command buffer is empty.

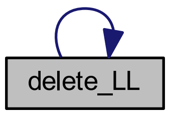
Parameters

void	

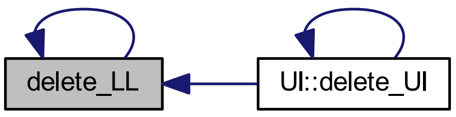
Returns

void

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.2.3 exec()

```
int exec (
             void )
```

(GLOBAL)executes the last command or the command buffer.

If the wait flag isnt set execute the command given by the bufferIndex. While there are still commands in the command buffer keep executing them.

if the waiting flag is set => return.

Possible inputs: "wacht", "repeat", "clearscherm", "lijn", "ellips", "rechthoek", "tekst", "bitmap" and "driehoek".

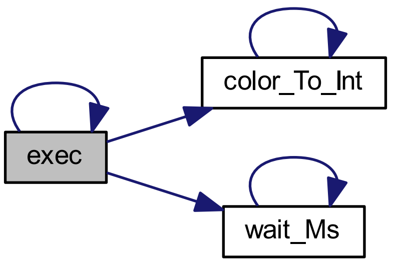
Parameters

void	

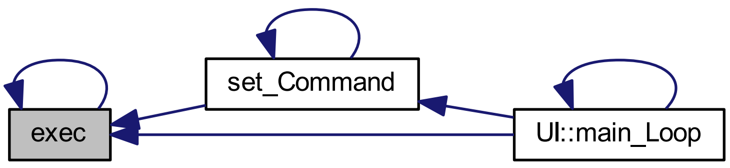
Returns

void

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.2.4 init_LL()

```
void init_LL (
             void  )
```

(GLOBAL)Initiate the LL.

Initiates the LogicLevel. All the flags are reset and a Vgascreen object is created. Everything is saved in the logic struct.

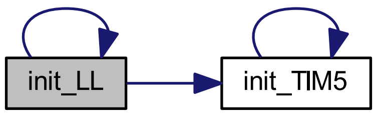
Parameters

void	

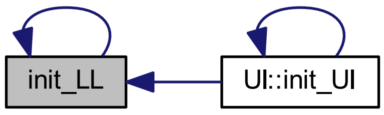
Returns

void

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.2.5 init_TIM5()

```
void init_TIM5 (
             void )
```

(LOCAL)Initiate TIM5 for the wait_Ms() function.

Sets the TIM5 and NVIC settings for interrupts on TIM5.

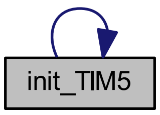
Parameters

void	

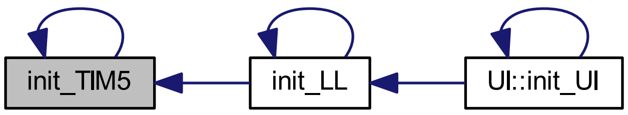
Returns

void

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.2.6 set_Command()

```
int set_Command (
              char * buf )
```

(GLOBAL)Sets the command struct.

Turns the input buffer into a command. It uses strtok_r to split the input at every ",".

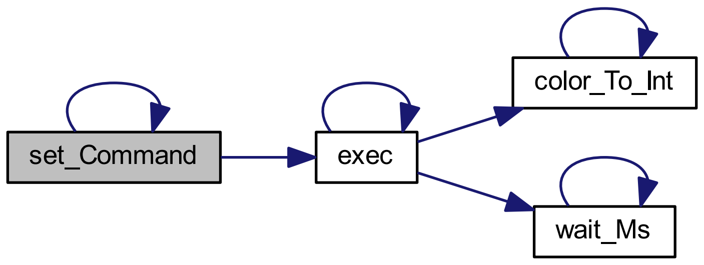
Parameters

char	∗buf the char buffer to set

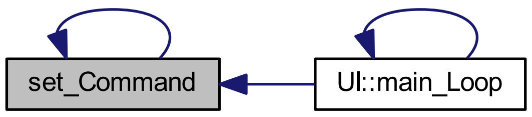
Returns

void

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.2.7 TIM5_IRQHandler()

```
void TIM5_IRQHandler (
             void  )
```

(GLOBAL) TIM5 interrupt handler for wait_Ms().

The TIM5_IRQh interrupts when the timer is triggered. The interrupt stops the read function of the IO-layer by calling IO::stop_Read() to let the UI call LL::exec(). The waiting FLAG is also cleared. Directly calling the LL::exec() function messes with the VGA output.

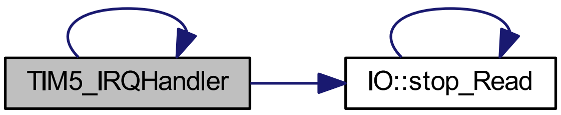
Parameters

void	

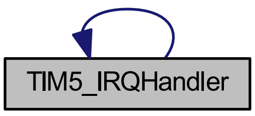
Returns

void

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.2.8 wait_Ms()

```
void wait_Ms (
              int ms )
```

(LOCAL)The wait function of the LL.

the function enables TIM5 and sets the prescale to give an interrupt at the chosen time. The logic.waiting FLAG is also set, so no further commands will be executed.

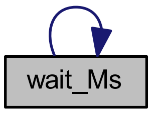
Parameters

int	ms, the time to wait

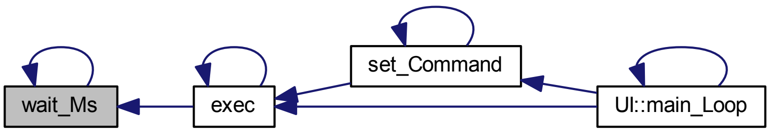
Returns

void

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.3 Variable Documentation

5.2.3.1 colors

```
const char* colors[COLORS]
```

Initial value:

```
=
{ "zwart", "blauw", "lichtblauw", "groen", "lichtgroen", "cyaan",
        "lichtcyaan", "rood", "lichtrood", "magenta", "lichtmagenta",
        "bruin", "geel", "grijs", "wit" }
```

List with all the color names.

5.2.3.2 logic

```
logic_t logic
```

Struct with the LL variables.

5.2.3.3 rgb

```
const int rgb[COLORS]
```

Initial value:

```
=
{ 0x00, 0x03, 0x0F, 0x1C, 0x0E, 0x1F, 0x7F, 0xE0, 0xED, 0xE3, 0xCE, 0x64,
        0xFC, 0x92, 0xFF }
```

List with all the color values.

5.3 UI Namespace Reference

Namespace UI.

Functions

- void delete_UI (void)

  (GLOBAL)Deletes the UI.
- void init_UI (void)

  (GLOBAL)Initiates the UI.
- void main_Loop (void)

  (Global)Starts the main UI loop.
- void write_Error (int err)

  (LOCAL)Write error.

5.3.1 Detailed Description

Namespace UI.

In the namespace UI are all the functions and variables concerning the User interface. The UI gets the user input from the IO layer and parses it to the Logic-layer. The LL may encounter an error and return an error code. The error text is send to the IO layer to be send back to the user if ERROR = on.

5.3.2 Function Documentation

5.3.2.1 delete_UI()

```
void delete_UI (
             void  )
```

(GLOBAL)Deletes the UI.

Deletes the UI-layer. Calls the LL::delete_LL() and the IO::delete_IO() functions.

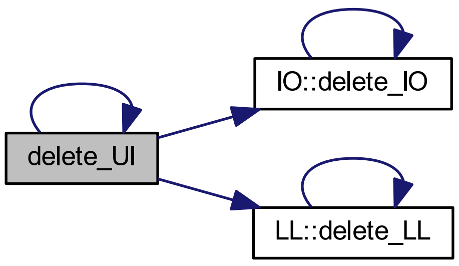
Parameters

void	

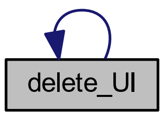
Returns

void

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.2.2 init_UI()

```
void init_UI (
             void  )
```

(GLOBAL)Initiates the UI.

Initializes the UI and initiates the logic layer and the IO-layer.

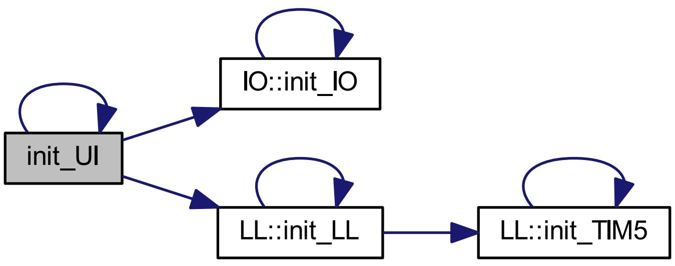
Parameters

void	

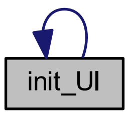
Returns

void

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.2.3 main_Loop()

```
void main_Loop (
             void  )
```

(Global)Starts the main UI loop.

Main UI loop. This is where the rest of the program happens. First the IO::read() function is called. If theres a new user input. LL::set_command() with the user input is called. If IO::read() returns empty call the LL::exec(). Errors from the LL will be printed using the UI::write_Error() function

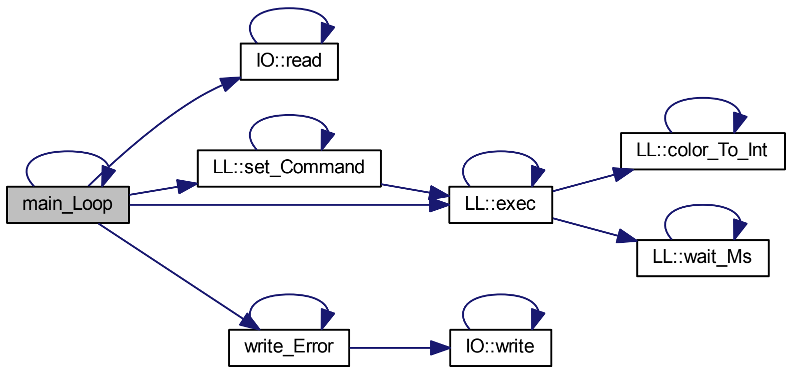
Parameters

void	

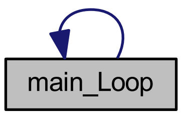
Returns

void

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.2.4 write_Error()

```
void write_Error (
             int err )
```

(LOCAL)Write error.

Writes the error message back to the user.

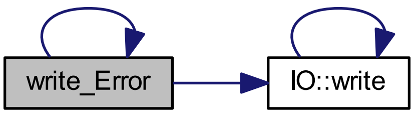
Parameters

void	

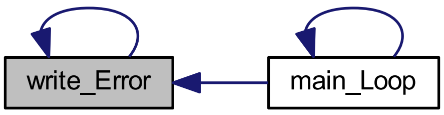
Returns

void

Here is the call graph for this function:



Here is the caller graph for this function:



6 Data Structure Documentation

6.1 command_t Struct Reference

Struct command_t for saving incoming commands.

```
#include "LogicLayer.h"
```

Data Fields

- char input1 [MAX_COLOR_LENGTH]
- char input2 [MAX_INT_LENGTH]
- char input3 [MAX_TEXT_LENGTH]
- char input4 [MAX_COLOR_LENGTH]
- char input5 [MAX_COLOR_LENGTH]
- char input6 [MAX_COLOR_LENGTH]
- char input7 [MAX_COLOR_LENGTH]
- char input8 [MAX_FILL_LENGTH]
- char type [MAX_COLOR_LENGTH]

6.1.1 Detailed Description

Struct command_t for saving incoming commands.

The command_t struct has 8 char arrays of different sizes. Different sizes are used to limit RAM usage.

6.1.2 Field Documentation

6.1.2.1 input1

```
char input1[MAX_COLOR_LENGTH]
```

First input of the command.

6.1.2.2 input2

```
char input2[MAX_INT_LENGTH]
```

Second input of the command.

6.1.2.3 input3

```
char input3[MAX_TEXT_LENGTH]
```

Third input of the command, this one is the largest for text.

6.1.2.4 input4

```
char input4[MAX_COLOR_LENGTH]
```

Fourth input of the command.

6.1.2.5 input5

```
char input5[MAX_COLOR_LENGTH]
```

Fifth input of the command.

6.1.2.6 input6

```
char input6[MAX_COLOR_LENGTH]
```

Sixth input of the command..

6.1.2.7 input7

```
char input7[MAX_COLOR_LENGTH]
```

Seventh input of the command.

6.1.2.8 input8

```
char input8[MAX_FILL_LENGTH]
```

eighth input of the command.

6.1.2.9 type

```
char type[MAX_COLOR_LENGTH]
```

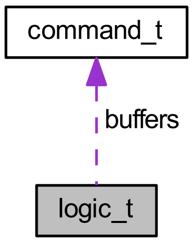
The type of command.

6.2 logic_t Struct Reference

logic_t struct for the different flags and buffer.

```
#include "LogicLayer.h"
```

Collaboration diagram for logic_t:



Data Fields

- volatile int bufferCnt
- volatile int bufferIndex
- command_t buffers [MAX_BUFFERS]
- Vgascreen screen
- volatile int waiting

6.2.1 Detailed Description

logic_t struct for the different flags and buffer.

The logic_t structs saves the different flags of the Logiclayer. The struct also saves the commands in a buffer.

6.2.2 Field Documentation

6.2.2.1 bufferCnt

```
volatile int bufferCnt
```

Amount of commands buffered.

6.2.2.2 bufferIndex

```
volatile int bufferIndex
```

Integer for the current command.

6.2.2.3 buffers

```
command_t buffers[MAX_BUFFERS]
```

Buffer for the commands.

6.2.2.4 screen

```
Vgascreen screen
```

The Vgascreen object.

6.2.2.5 waiting

```
volatile int waiting
```

The waiting FLAG.

Index