stm32f4-discovery, 320, 240 VGA

User Manual  
Deze handleiding bevat informatie over het ontwerp van de VGA applicatie en de functies die gebruikt kunnen worden met de EE-API library.

Matthijs Daggelders & Niek Ratering Arntz

31-05-2018

# Inleiding (beschrijving)

De stm32f4-discovery 320x240 VGA is een C/C++ project dat draait op de stm32f4-discovery en stuurt een 320x240 VGA scherm aan. Door verschillende input aan de stm32f4-discovery board via UART te geven. Worden verschillende dingen op het scherm getekend. Zo kan er tekst in verschillende stijlen, lijnen, ellipsen en andere vormen op het VGA scherm getekend worden. Dit kan per input gedaan worden maar er kan ook een lus gecreëerd worden. Een lus kan gemaakt worden door eerst een wait opdracht te sturen en vervolgens de opdrachten die herhaald moeten worden afsluiten met een *“repeat” opdracht.* In de lus worden de input achterelkaar uitgevoerd en op het einde wordt er weer bij het begin begonnen. Om de loop vervolgens te verlaten kan elke willekeurige opdracht gestuurd worden.

Inhoudsopgave

[Inleiding(beschrijving) 3](#_Toc515542461)

[Ontwerp 3-lagen 5](#_Toc515542462)

[Toelichting 3-lagen model 6](#_Toc515542463)

[EE-API.lib 7](#_Toc515542464)

[Doxygen 8](#_Toc515542465)

[Doxygen 3-lagen model 8](#_Toc515542466)

[Doxygen EE-API.LIB 8](#_Toc515542467)

[Functie lijst 8](#_Toc515542468)

[draw\_line() 8](#_Toc515542469)

[draw\_ellipse() 9](#_Toc515542470)

[draw\_rectangle() 10](#_Toc515542471)

[draw\_triangle() 11](#_Toc515542472)

[draw\_text() 12](#_Toc515542473)

[draw\_bitmap() 13](#_Toc515542474)

[clear\_screen() 14](#_Toc515542475)

[write() 15](#_Toc515542476)

[read() 16](#_Toc515542477)

[stop\_Read() 17](#_Toc515542478)

[init\_UART2() 17](#_Toc515542479)

[init\_IDLE\_Line() 18](#_Toc515542480)

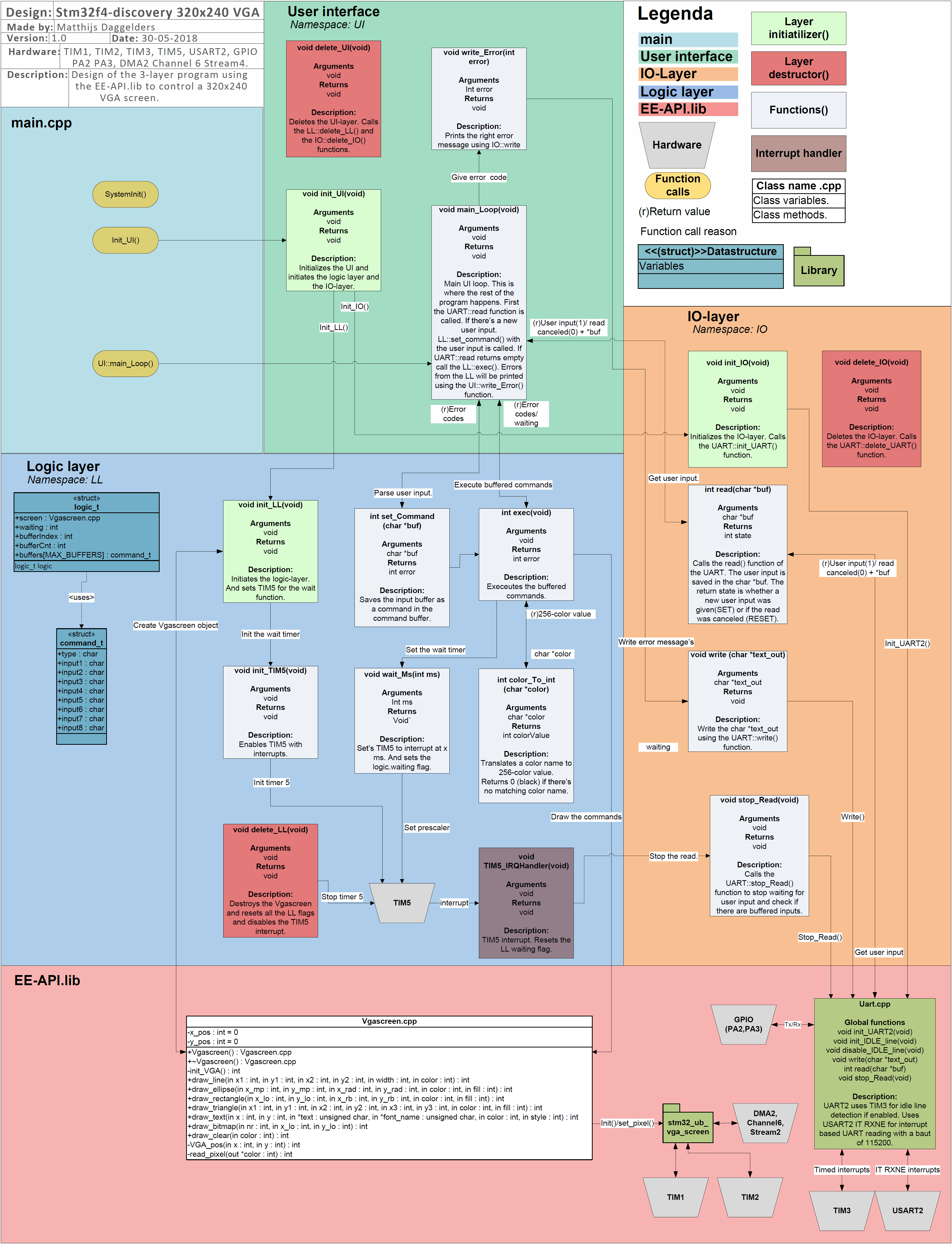
[disable\_IDLE\_Line() 18](#_Toc515542481)

[delete\_UART() 19](#_Toc515542482)

[TIM3\_IRQHandler() 19](#_Toc515542483)

[USART2\_IRQHandler() 20](#_Toc515542484)

# Ontwerp 3-lagen

Hieronder is het 3-lagen model van de software weergegeven.  


Figuur . Ontwerp 3-lagen model met de EE-API.lib.

## Toelichting 3-lagen model

In figuur 1 is het ontwerp van het 3-lagen model met de EE-API.lib weergegeven. Het onderste lichtrode vlak is de EE-API.lib. Daarboven zijn de 3-lagen van het programma en de main.cpp weergegeven. De drie lagen van het model zijn de:

* **User interface (UI)**De user interface bepaalt wat de gebruiker te zien krijgt en is de basis van het programma. De UI heeft een *main\_Loop()* functie waarin de rest van het programma werkt. Eerst wordt de input van de gebruiker opgevraagd via de IO-layer *(IO::read()).* Als er een input binnen is gekomen wordt deze doorgestuurd naar de LL. De LL verwerkt de input van de gebruiker en geeft een error code terug als de input niet herkent wordt of als de input buffer vol zit. De *write\_Error()* functie vertaald de error code naar een output string en stuurt deze door naar de IO-layer. De IO-layer stuurt het op zijn beurt door naar de UART, die de error melding op de terminal weergeeft.

Het is ook mogelijk dat de *IO::stop\_Read()* wordt aangeroepen. Wanneer dit gebeurd stopt het programma met wachten op gebruiker input en wordt in de UI de *LL::exec()* aangeroepen om te kijken of er gebufferde opdrachten( commands) uitgevoerd kunnen worden.

* **Input/output-layer (IO-layer)**

De IO-layer functioneert als de laag tussen het programma en de UART library functionaliteit. Als de LL-layer of de UI iets met de UART willen doen gaat dit door de IO-layer heen. Doordat alles door de IO-layer heen gaat. Hoeft alleen deze layer aangepast te worden als een andere input bron gebruikt wordt (bijvoorbeeld I2C). De UI en de LL kunnen die IO-layer op dezelfde functies aanspreken alleen wat erna gebeurd veranderd. Hierbij moet rekening gehouden worden met het programma verloop.

Als de *UART::read()* wordt aangeroepen blijft het programma net zolang wachten tot er een user input gegeven is of de *UART::stop\_Read()* wordt aangeroepen. Deze functies worden aangeroepen met dezelfde functienaam alleen dan in de IO namespace.

* **Logic-Layer (LL)**

De LL is waar de user input wordt opgesplitst in losse woorden. Dit gebeurd in de *set\_command()* functie. De opdracht wordt opgeslagen in *logic.buffers[].* Met de LL::exec() functie kan de huidige command of alle gebufferde commands uitgevoerd worden.

De LL::exec() functie bepaald welk VGAscreen functie aangeroepen moet worden. En vertaald de command naar de juiste format voor de VGAscreen functie.

De wait\_Ms() functie van de LL zet de waiting timer hoog en zet de prescaler van de timer zo in dat het gewenste aantal ms gewacht wordt. Tijdens het wachten kunnen er wel commands in de buffer worden opgeslagen maar de *exec()* command voert niks uit en keert terug naar de UI om vervolgens weer te wachten op de gebruiker input. Als timer 5 een interrupt geeft, wordt de *IO::stop\_Read()* functie opgeroepen. Het stoppen van de IO::read() zorgt ervoor dat de LL::exec() functie wordt aangeroepen in de *UI::main\_Loop().* De *LL::exec()* voert alle commands uit de buffer uit totdat er een nieuw “wait” command gegeven is. Als het laatste command in de buffer “repeat” is dan worden alle commands die opgeslagen zijn herhaald. Dit gebeurd net zolang tot er een nieuwe input gegeven is.

De logic-layer staat als een mediator tussen het programma en de VGA functies. Hierdoor als er iets veranderd in de VGAscreen klasse hoeft alleen de LL aangepast te worden. Zelfde geld als bijvoorbeeld de UI veranderd zal de LL het zelfde blijven. Dit komt doordat de nieuwe UI zonder problemen de LL functies aan kan spreken, zonder wijzigingen aan de LL kant.

## EE-API.lib

# Doxygen

## Doxygen 3-lagen model

Zie pdfbijlagen “DOXYGEN STM32F4-Discovery, 320x240 VGA V1.0.pdf”.



## Doxygen EE-API.LIB

Zie pdfbijlagen “EE-API.pdf”.



# Functie lijst

## draw\_line()

**draw\_line**(**int** x1, **int** y1, **int** x2, **int** y2, **int** width, **int** color);

|  |  |  |
| --- | --- | --- |
| ***File*** | ***Called from*** | ***Code enabled by*** |
| Vgascreen.cpp | task | init\_VGA(); |

draw\_line() wordt gebruikt om lijnen te tekenen op een VGA scherm.

**Argumenten**

X1 x coördinaat van het beginpunt van de lijn.

Y1 y coördinaat van het beginpunt van de lijn.

X2 x coördinaat van het eindpunt van de lijn.

Y2 y coördinaat van het eindpunt van de lijn.

Width Dikte van de lijn in pixels.

Color Kleur van de lijn (1 t/m 255).

**Return waardes**

Returns 0 bij succesvol uitvoeren van de functie.

**Opmerkingen/Waarschuwing**

De x en y coördinaten van de punten moeten binnen het aantal beeldpunten liggen van het VGA scherm (320 breed, 240 hoog).

**Voorbeeld**

draw\_line(20,20,,40,50,1,0xE0);

## draw\_ellipse()

**draw\_ellipse**(**int** x\_mp, **int** y\_mp, **int** x\_rad, **int** y\_rad, **int** color, **int** fill);

|  |  |  |
| --- | --- | --- |
| ***File*** | ***Called from*** | ***Code enabled by*** |
| Vgascreen.cpp | task | init\_VGA(); |

draw\_ellipse() wordt gebruikt om ellipsen te tekenen op een VGA scherm.

**Argumenten**

x\_mp x coördinaat van het middelpunt van de ellips.

y\_mp y coördinaat van het middelpunt van de ellips.

x\_rad Breedte van de ellips.

y\_rad Hoogte van de ellips.

color Kleur van de ellips (1 t/m 255).

fill 1 om de ellips te vullen (optioneel).

**Return waardes**

Returns 0 bij succesvol uitvoeren van de functie.

**Opmerkingen/Waarschuwing**

De x en y coördinaten van de punten moeten binnen het aantal beeldpunten liggen van het VGA scherm (320 breed, 240 hoog).

**Voorbeeld**

draw\_ellipse(100,150,,40,65,0xE0,0);

## draw\_rectangle()

**draw\_rectangle**(**int** x\_lo, **int** y\_lo, **int** x\_rb, **int** y\_rb, **int** color, **int** fill);

|  |  |  |
| --- | --- | --- |
| ***File*** | ***Called from*** | ***Code enabled by*** |
| Vgascreen.cpp | task | init\_VGA(); |

draw\_rectangle() wordt gebruikt om rechthoeken te tekenen op een VGA scherm.

**Argumenten**

x\_lo x coördinaat van de linker onderhoek van de rechthoek.

y\_lo y coördinaat van de linker onderhoek van de rechthoek.

x\_rb x coördinaat van de rechter bovenhoek van de rechthoek.

y\_rb y coördinaat van de rechter bovenhoek van de rechthoek.

color Kleur van de rechthoek (1 t/m 255).

fill 1 om de rechthoek te vullen (optioneel).

**Return waardes**

Returns 0 bij succesvol uitvoeren van de functie.

**Opmerkingen/Waarschuwing**

De x en y coördinaten van de punten moeten binnen het aantal beeldpunten liggen van het VGA scherm (320 breed, 240 hoog).

**Voorbeeld**

draw\_rectangle(50,150,,100,50,0xE0,0);

## draw\_triangle()

**draw\_triangle**(**int** x1, **int** y1, **int** x2, **int** y2, **int** x3, **int** y3, **int** color, **int** fill);

|  |  |  |
| --- | --- | --- |
| ***File*** | ***Called from*** | ***Code enabled by*** |
| Vgascreen.cpp | task | init\_VGA(); |

draw\_triangle() wordt gebruikt om driehoeken te tekenen op een VGA scherm.

**Argumenten**

x1 x coördinaat van hoek 1 van de driehoek.

y1 y coördinaat van hoek 1 van de driehoek.

x2 x coördinaat van hoek 2 van de driehoek.

y2 y coördinaat van hoek 2 van de driehoek.

x3 x coördinaat van hoek 3 van de driehoek.

y3 y coördinaat van hoek 3 van de driehoek.

color Kleur van de driehoek (1 t/m 255).

fill 1 om de driehoek te vullen (optioneel).

**Return waardes**

Returns 0 bij succesvol uitvoeren van de functie.

**Opmerkingen/Waarschuwing**

De x en y coördinaten van de punten moeten binnen het aantal beeldpunten liggen van het VGA scherm (320 breed, 240 hoog).

**Voorbeeld**

draw\_triangle(20,20,90,35,40,60,0xE0,0);

## draw\_text()

**draw\_text**(**int** x, **int** y, **const** **char** \*str, **int** color, **const** **char**\* style, **int** fontNr);

|  |  |  |
| --- | --- | --- |
| ***File*** | ***Called from*** | ***Code enabled by*** |
| Vgascreen.cpp | task | init\_VGA(); |

draw\_text() wordt gebruikt om tekst te schrijven op een VGA scherm.

**Argumenten**

x x coördinaat vanaf waar de tekst wordt geschreven.

y y coördinaat vanaf waar de tekst wordt geschreven.

str In te voeren tekst.

color Kleur van de tekst (1 t/m 255).

style 3 soorten stijlen: “*norm*” 🡪 normale letters,

“*cursief*” 🡪 cursieve tekst,

“*vet*” 🡪 dikgedrukte tekst.

fontNr 2 soorten lettertypes: 1 voor font1 en 2 voor font2

**Return waardes**

Returns 0 bij succesvol uitvoeren van de functie.

**Opmerkingen/Waarschuwing**

De x en y coördinaten van de punten moeten binnen het aantal beeldpunten liggen van het VGA scherm (320 breed, 240 hoog).

**Voorbeeld**

draw\_text(30,100,“the quick brown fox jumps over the lazy dog”,0xE0,”cursief”,1);

## draw\_bitmap()

**draw\_bitmap**(**int** nr, **int** x\_lo, **int** y\_lo);

|  |  |  |
| --- | --- | --- |
| ***File*** | ***Called from*** | ***Code enabled by*** |
| Vgascreen.cpp | task | init\_VGA(); |

draw\_bitmap() wordt gebruikt om bitmaps te tekenen op een VGA scherm. De bitmaps zijn elk 32 pixels hoog en 32 pixels breed.

**Argumenten**

nr Het nummer van de bitmap. Er zijn 6 verschillende bitmaps beschikbaar:

0 🡪 blije smiley

1 🡪 boze smiley

2 🡪 pijl naar boven wijzend

3 🡪pijl naar beneden wijzend

4 🡪pijl naar rechts wijzend

5 🡪pijl naar links wijzend

x\_lo x coördinaat van de linker onderhoek van de bitmap.

y\_lo y coördinaat van de linker onderhoek van de bitmap.

**Return waardes**

Returns 0 bij succesvol uitvoeren van de functie.

**Opmerkingen/Waarschuwing**

De x en y coördinaten van de punten moeten binnen het aantal beeldpunten liggen van het VGA scherm (320 breed, 240 hoog).

**Voorbeeld**

draw\_bitmap(2,150,100);

## clear\_screen()

**clear\_screen**(**int** color);

|  |  |  |
| --- | --- | --- |
| ***File*** | ***Called from*** | ***Code enabled by*** |
| Vgascreen.cpp | task | init\_VGA(); |

clearscreen() wordt gebruikt om het beeld leeg te maken en op te vullen met een kleur naar keuze.

**Argumenten**

color De kleur die het beeldscherm moet aannemen.

**Return waardes**

Returns 0 bij succesvol uitvoeren van de functie.

**Opmerkingen/Waarschuwing**

N/A

**Voorbeeld**

clear\_screen(0xE0);

## write()

**write**(**char** \*text\_out);

|  |  |  |
| --- | --- | --- |
| ***File*** | ***Called from*** | ***Code enabled by*** |
| Uart.cpp | task | init\_UART2(); |

write() wordt gebruikt tekst te schrijven naar de UART2. De functie print een string per karakter met de put\_Char() functie.

**Argumenten**

tekst\_out De tekst die geprint moet worden.

**Return waardes**

Void

**Opmerkingen/Waarschuwing**

N/A

## read()

**read**(**char** \*buf);

|  |  |  |
| --- | --- | --- |
| ***File*** | ***Called from*** | ***Code enabled by*** |
| Uart.cpp | task | init\_UART2(); |

read() wordt gebruikt om gebruikersinput te lezen van de USART2.

**Argumenten**

buf Een buffer met de gebruikersinput.

**Return waardes**

Int error

**Opmerkingen/Waarschuwing**

N/A

## stop\_Read()

**stop\_read**(void);

|  |  |  |
| --- | --- | --- |
| ***File*** | ***Called from*** | ***Code enabled by*** |
| Uart.cpp | task | init\_UART2(); |

read() stopt de read functie wanneer er geen userinput wordt gegeven.

**Argumenten**

N/A

**Return waardes**

void

**Opmerkingen/Waarschuwing**

N/A

## init\_UART2()

**init\_UART2**(void);

|  |  |  |
| --- | --- | --- |
| ***File*** | ***Called from*** | ***Code enabled by*** |
| Uart.cpp | task | init\_UART2(); |

init\_UART2() initieert de UART componenten. Stelt de UART\_T struct in en initieert de RCC,GPIO,USART en de NVIC

**Argumenten**

N/A

**Return waardes**

void

**Opmerkingen/Waarschuwing**

N/A

## init\_IDLE\_Line()

**init\_LDLE\_Line**(void);

|  |  |  |
| --- | --- | --- |
| ***File*** | ***Called from*** | ***Code enabled by*** |
| Uart.cpp | task | init\_UART2(); |

init\_LDLE\_Line() initieert de IDLE line detection voor UART2. Setup van TIM3 en enables interrupts.

**Argumenten**

N/A

**Return waardes**

void

**Opmerkingen/Waarschuwing**

N/A

## disable\_IDLE\_Line()

**disable\_LDLE\_Line**(void);

|  |  |  |
| --- | --- | --- |
| ***File*** | ***Called from*** | ***Code enabled by*** |
| Uart.cpp | task | init\_UART2(); |

disable\_LDLE\_Line() disabled de IDLE line detection voor UART2. Disables TIM3 en TIM3 interrupts.

**Argumenten**

N/A

**Return waardes**

void

**Opmerkingen/Waarschuwing**

N/A

## delete\_UART()

**delete\_UART**(void);

|  |  |  |
| --- | --- | --- |
| ***File*** | ***Called from*** | ***Code enabled by*** |
| Uart.cpp | task | init\_UART2(); |

delete\_UART() stopt UART en TIM3.

**Argumenten**

N/A

**Return waardes**

void

**Opmerkingen/Waarschuwing**

N/A

## TIM3\_IRQHandler()

**TIM3\_IRQHandler**(void);

|  |  |  |
| --- | --- | --- |
| ***File*** | ***Called from*** | ***Code enabled by*** |
| Uart.cpp | task | init\_UART2(); |

TIM3\_IRQHandler() roept de UART::stop\_Read() functie om door te gaan met het uitvoeren van de buffer.

**Argumenten**

N/A

**Return waardes**

void

**Opmerkingen/Waarschuwing**

N/A

## USART2\_IRQHandler()

**USART2\_IRQhandler**(void);

|  |  |  |
| --- | --- | --- |
| ***File*** | ***Called from*** | ***Code enabled by*** |
| Uart.cpp | task | init\_UART2(); |

USART2\_IRQHandler() vult de input buffer en zet de LL::logic\_t.bReady flag waneer deze klaar is.

**Argumenten**

N/A

**Return waardes**

void

**Opmerkingen/Waarschuwing**

N/A