

COPYRIGHT

Copyright © 2020 Stellenbosch University
All rights reserved

DISCLAIMER

This content is provided without warranty or representation of any kind. The use of the content is entirely at your own risk and Stellenbosch University (SU) will have no liability directly or indirectly as a result of this content.

The content must not be assumed to provide complete coverage of the particular study material. Content may be removed or changed without notice.

The video is of a recording with very limited post-recording editing. The video is intended for use only by SU students enrolled in the particular module.



UNIVERSITEIT
iYUNIVESITHI
STELLENBOSCH
UNIVERSITY



forward together · saam vorentoe · masiye phambili

Computer Systems / Rekenaarstelsels 245 - 2020

Lecture 6a

Introduction to Development Board and IDE

Inleiding tot Ontwikkelsbord en IDE

Dr Rensu Theart & Dr Lourens Visagie

Lecture Overview

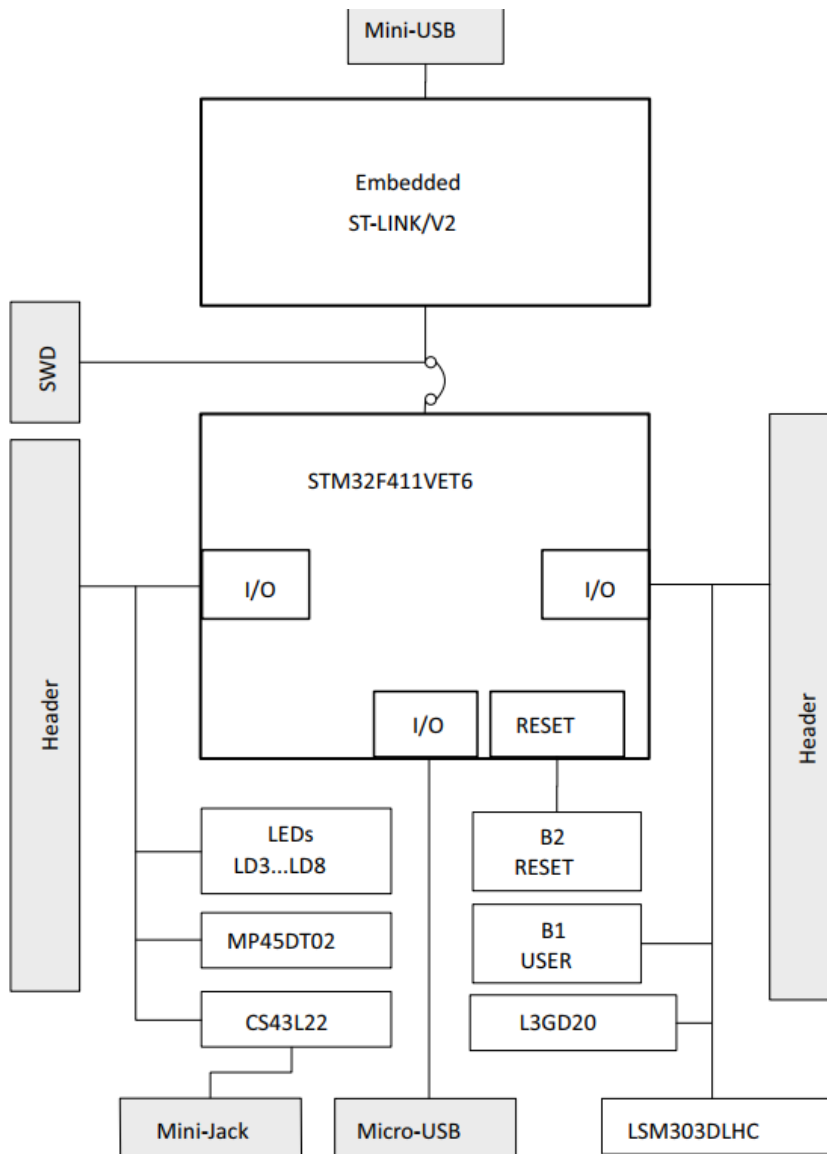
- Development board
- Emulator
- STM32CubeIDE

Development board / Ontwikkelingsbord

- Development board for the **STM32F411VET6** microcontroller (STM32F411 Discovery Kit)
- Board and microcontroller manufactured by ST Microelectronics
- In the microcontroller:
 - **ARM Cortex-M4** with floating point unit
 - 512 kb flash memory (program memory)
 - 128 kb SRAM
- Other stuff on the board:
 - 3D MEMS gyroscope, linear accelerometer, magnetic field sensor
 - MEMS microphone, audio digital-to-analog converter (DAC) and amplifier
 - LEDs, pushbuttons



Development board / Ontwikkelingsbord



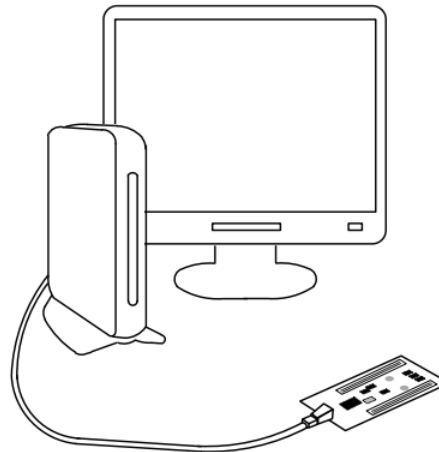
- B1 and B2 : buttons
- LD3..LD8 : LEDs
- MP45DT02: MEMS microphone
- CS43L22: Audio DAC
- L3GD020: 3-axis gyroscope
- LSM303DLHC: Linear accelerometer and magnetic field sensor

Power and programming / Kragen programming

- Board gets **power** from USB (from the PC that it is connected to)
- **Programming** also happens through USB
- Binary program has to be placed in flash memory – processor will start executing instructions from address 0.
- ST-Link: In-circuit debugger and programmer – included on the development board (interfaces using USB)



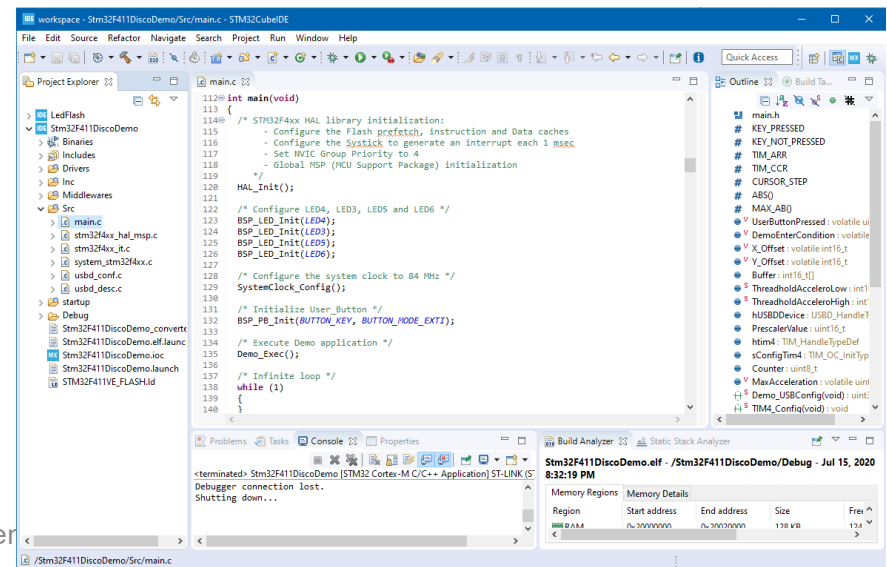
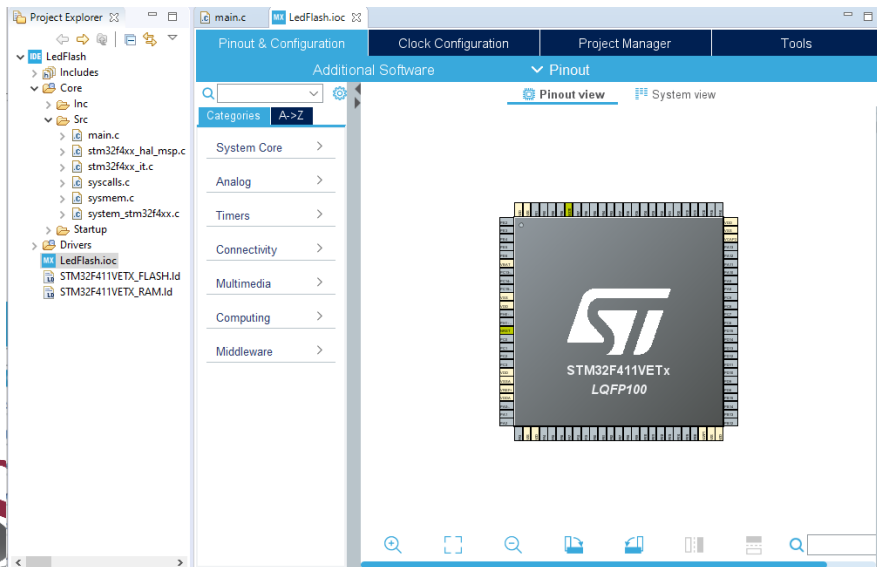
Do not remove the jumpers!!
(default selection for programming)



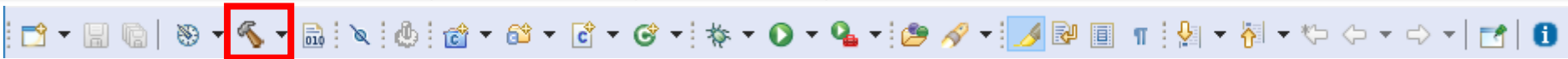
0xFFFF FFFF	512-Mbyte block 7 Cortex-M4's internal peripherals
0xE000 0000 0xDFFF FFFF	512-Mbyte block 6 Not used
0xC000 0000 0xBFFF FFFF	Reserved
0x6000 0000 0x5FFF FFFF	512-Mbyte block 2 Peripherals
0x4000 0000 0x3FFF FFFF	512-Mbyte block 1 SRAM
0x2000 0000 0x1FFF FFFF	512-Mbyte block 0 Code
0x0000 0000	

Development Environment / Ontwikkelings Omgewing

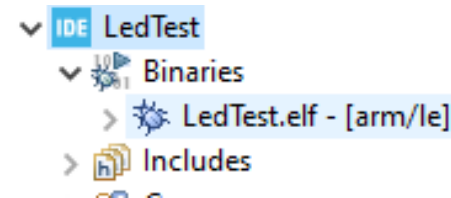
- IDE : Integrated Development Environment
- A source code editor, build automation and debugger
- **STM32CubeIDE** is an advanced C/C++ development platform with peripheral configuration, code generation, code compilation, and debug features for STM32 microcontrollers and microprocessors.
- It is based on the ECLIPSE™ framework and GCC toolchain for the development, and GDB for the debugging.



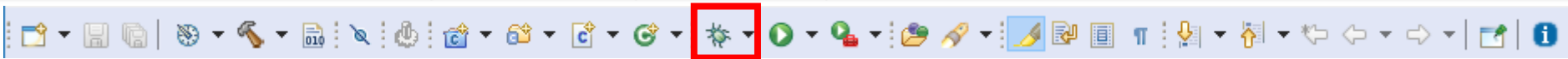
Development Environment / Ontwikkelings Omgewing



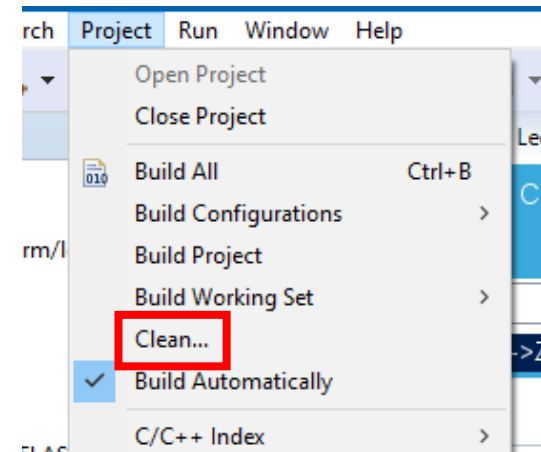
- What does it mean to '**Build**' project :
 1. Compile all (.c) source files to object files (**compiler**)
 2. Assemble all (.s) assembly files to object files (**assembler**)
 3. Link all object files (.o) to a binary executable file (.elf) (**linker**)
- The file with the 'elf' extension (elf = Extensible Linking Format) contains the program code that will be programmed into flash memory.
- The compiler, assembler and linker is called the **toolchain**.
- arm-none-eabi-gcc is the toolchain we use. This toolchain targets the ARM architecture, and does not target an operating system (i.e. targets a "bare metal" system).
- **Cross-compile**: is a **compiler** capable of creating executable code for a platform other than the one on which the compiler is running – typical of embedded systems.



Development Environment / Ontwikkelings Omgewing

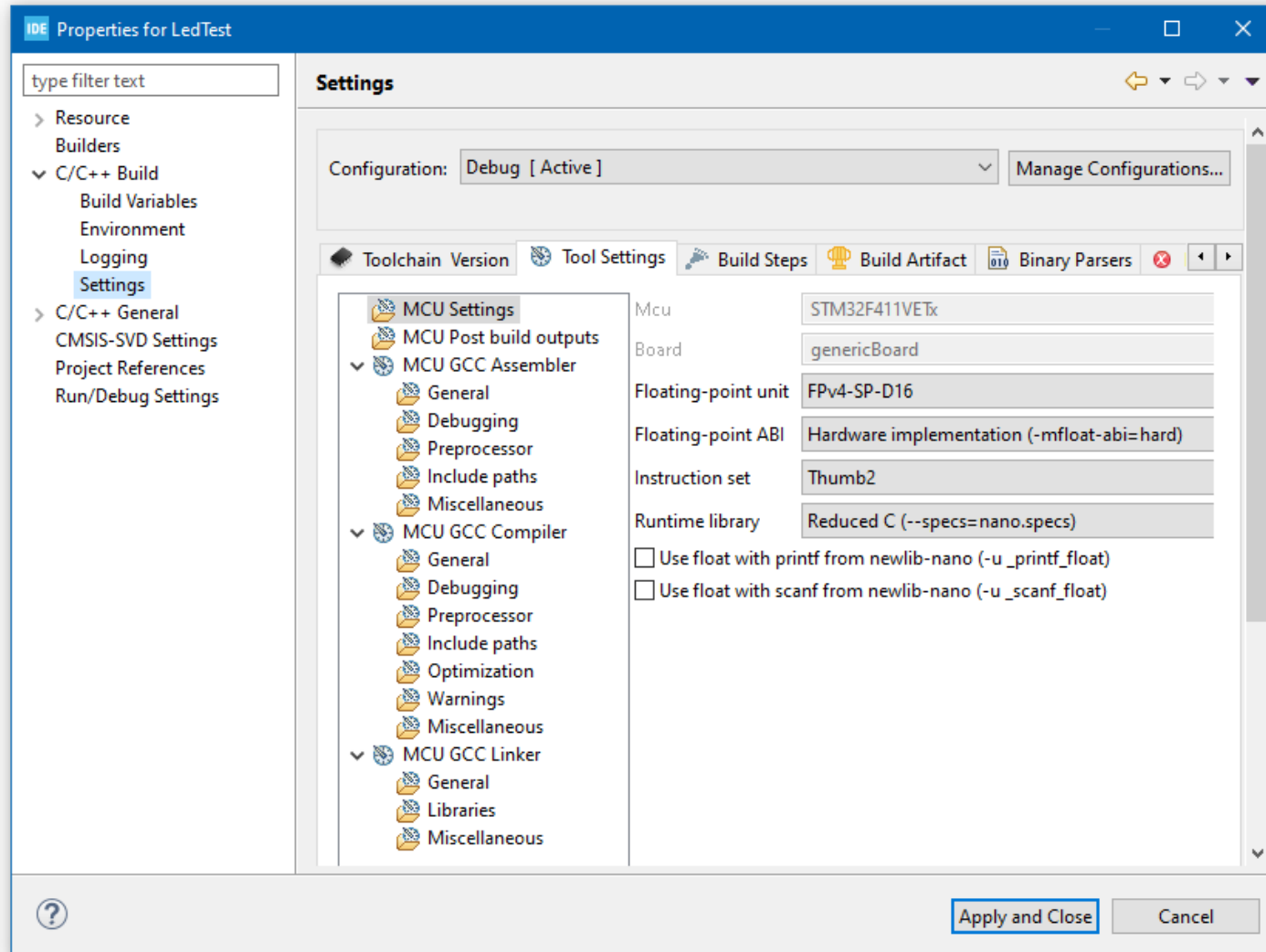


- What does it mean to '**Clean**' project :
 - 'Build' is usually incremental. IDE is smart and only recompiles files that have changed,
 - Remove all the .o files and .elf file. (Next time you press 'Build', everything will be recompiled from scratch)
- What does it mean to '**Debug**':
 - Pressing this button will first cause program code to be loaded into the microcontroller flash memory.
 - Program will start to execute (processor will start fetching decoding and executing instructions)
 - Through a debugging interface, allow the IDE to 'pause' the processor at specific points in the program (**breakpoint**)
 - Step through program code, line-for-line
 - Inspect (or modify) variable values, register and memory contents while program is halted.

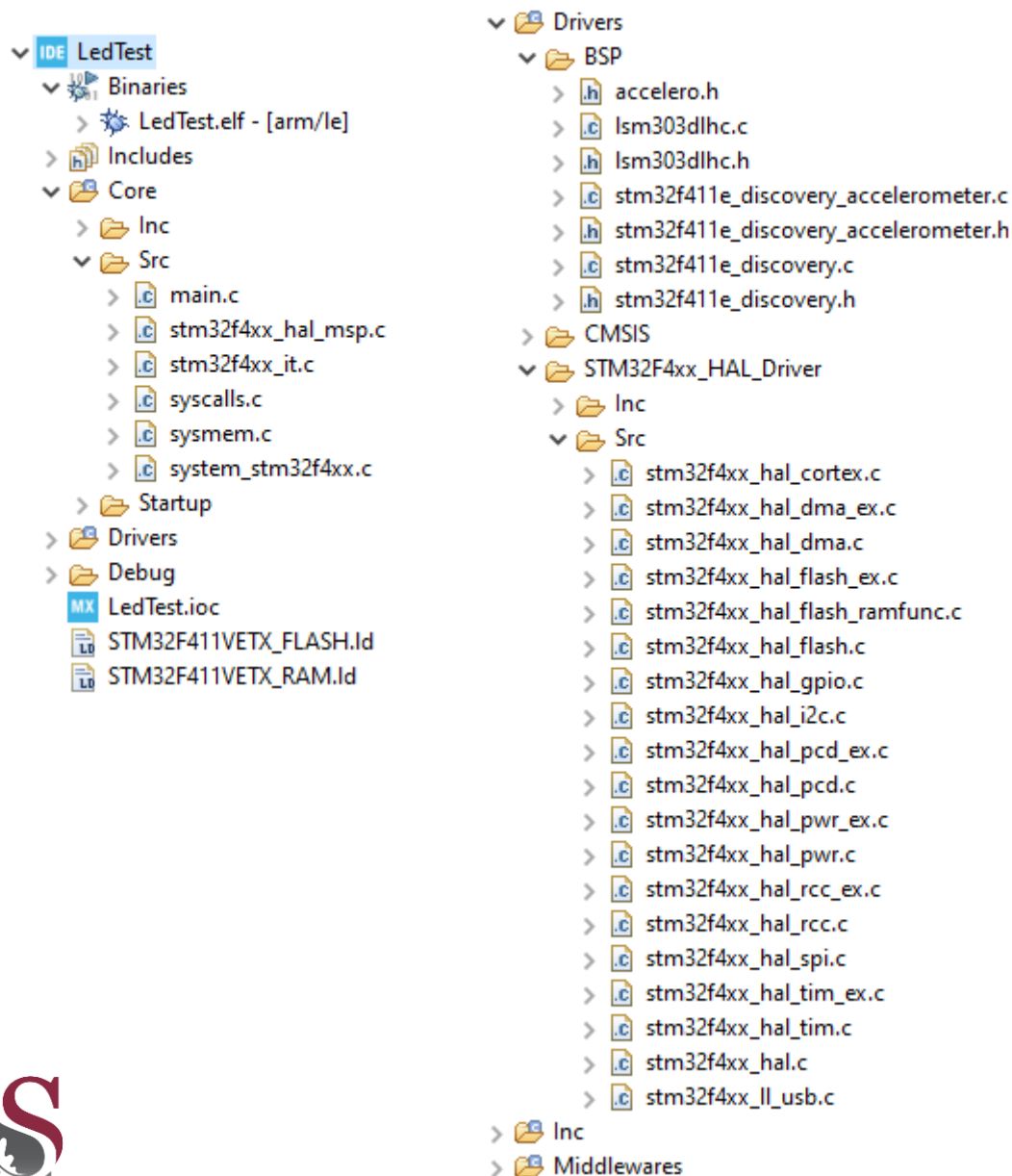


Project properties/ Projekterstellungen

- Right-click on project, select 'Properties'



Project source files / Projek bron lêers

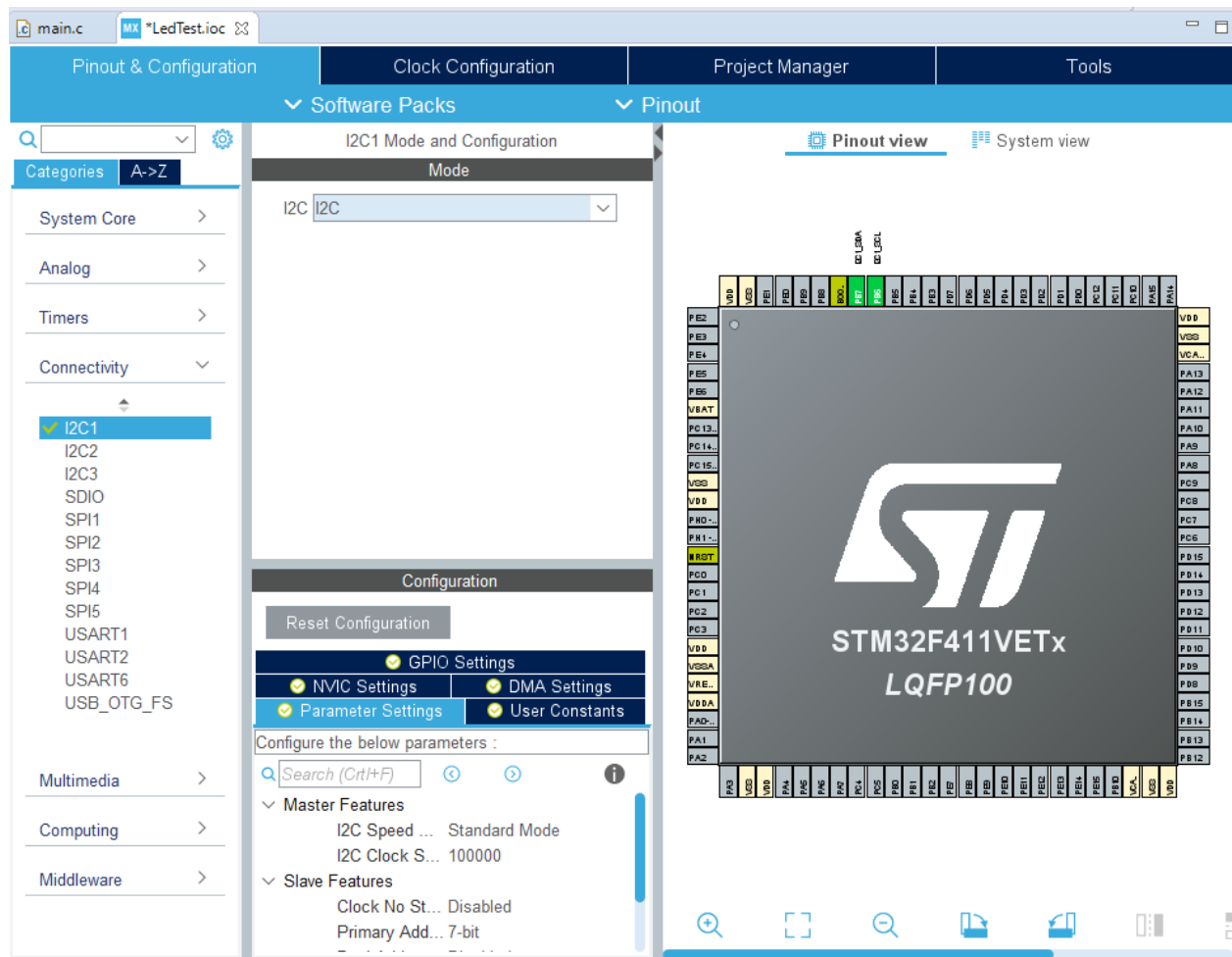
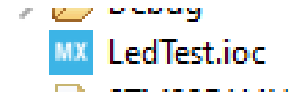


- Lots of source files...
- **BSP** = Board Support Package (driver code for peripherals on the development board)
- **HAL** = Hardware Abstraction Layer (driver code for peripherals on the microcontroller)
- **Middelwares**: FreeRTOS (operating system), JPG library, etc.



Creating project files / Skep van projek lêers

- Easily set up a project file with graphical user interface.
- Access again by opening the .ioc file:



Resources on SunLearn / Hulpmiddels op SunLearn

- Documentation for the development board and microcontroller (on SunLearn, under 'Resources/Hulpmiddels')
 - Datasheet - hardware
 - Reference manual – memory layout and registers
 - Programming manual – all the supported instructions for STM32F4
- Also
 - Summary of ARM instruction set
 - ARMv7 Architecture Reference Manual
- STM32CubeIDE
 - Installation files
 - Installation and setup guides



HAL Reference manual

- Refer to the manual on SunLearn for writing to and reading from GPIOs and other peripherals.



UM1725 User Manual

Description of STM32F4 HAL and LL drivers

Introduction

STM32Cube™ is STMicroelectronics's original initiative to ease developers' life by reducing development efforts, time and cost. STM32Cube™ covers the STM32 portfolio.

STM32Cube™ Version 1.x includes:

- The STM32CubeMX, a graphical software configuration tool that allows generating C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeF4 for STM32F4 Series)
 - The STM32Cube hardware abstraction layer (HAL), an STM32 abstraction layer embedded software, ensuring maximized portability across the STM32 portfolio
 - The Low Layer APIs (LL) offering a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. The LL APIs are available only for a set of peripherals
 - A consistent set of middleware components such as RTOS, USB, TCP/IP, Graphics
 - All embedded software utilities coming with a full set of examples.

The HAL driver layer provides a generic multi instance simple set of APIs (application programming interfaces) to interact with the upper layer (application, libraries and stacks). It is composed of generic



HAL Reference manual

HAL_GPIO_WritePin

Function name `void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)`

Function description Sets or clears the selected data port bit.

Parameters

- **GPIOx:** where x can be (A..K) to select the GPIO peripheral for STM32F429X device or x can be (A..I) to select the GPIO peripheral for STM32F40XX and STM32F427X devices.
- **GPIO_Pin:** specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15).
- **PinState:** specifies the value to be written to the selected bit. This parameter can be one of the GPIO_PinState enum values:
 - GPIO_PIN_RESET: to clear the port pin
 - GPIO_PIN_SET: to set the port pin

Return values • **None:**

Notes

- This function uses GPIOx_BSRR register to allow atomic read/modify accesses. In this way, there is no risk of an IRQ occurring between the read and the modify access.

- Example:

```
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_SET);
```



HAL Reference manual

- When you start typing the name and press Ctrl+Space you can see “Template Proposals” which give you all the possible values.

```
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_, GPIO_PIN_SET);
```



- GPIO_PIN_RESET
- GPIO_PIN_SET
- # GPIO_PIN_0
- # GPIO_PIN_1
- # GPIO_PIN_10
- # GPIO_PIN_11
- # GPIO_PIN_12
- # GPIO_PIN_13
- # GPIO_PIN_14
- # GPIO_PIN_15
- # GPIO_PIN_2
- # GPIO_PIN_3

Press 'Ctrl+Space' to show Template Proposals

- You must still understand what you are doing, since this is only a list of constants, and not the correct value for the function argument.



UNIVERSITEIT
iYUNIVESITHI
STELLENBOSCH
UNIVERSITY



forward together · saam vorentoe · masiye phambili

Computer Systems / Rekenaarstelsels 245 - 2020

Lecture 6b

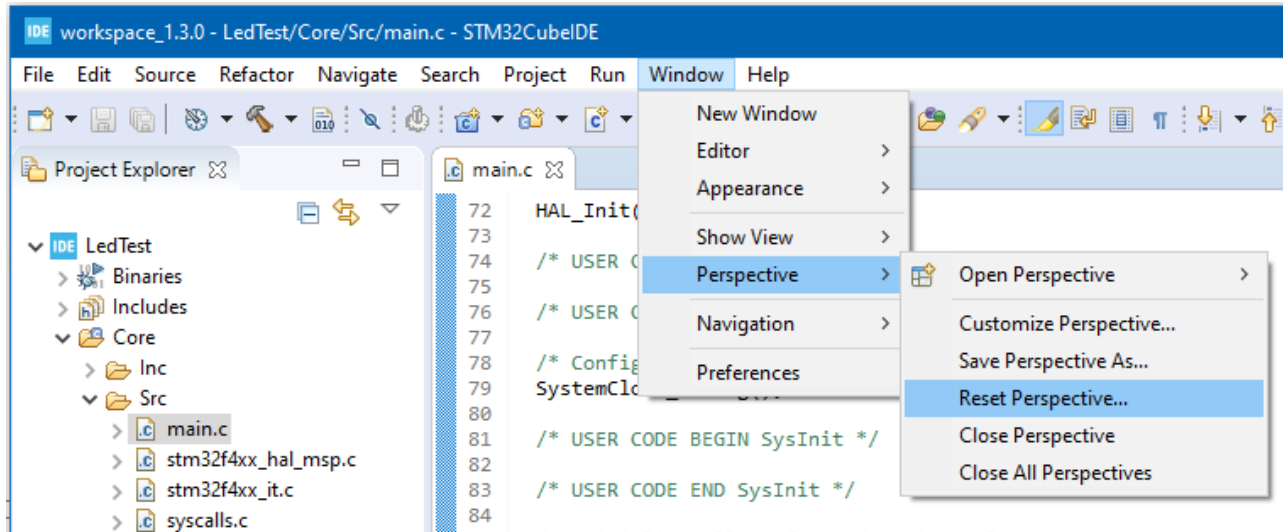
Programming tips

Programmerings wenke

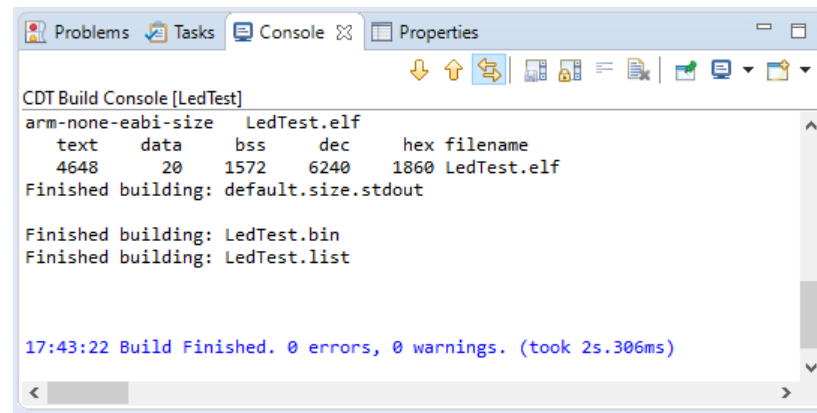
Dr Rensu Theart & Dr Lourens Visagie

Hints/ Wenke

1. Reset the IDE 'perspective' – to restore all the windows to their default locations



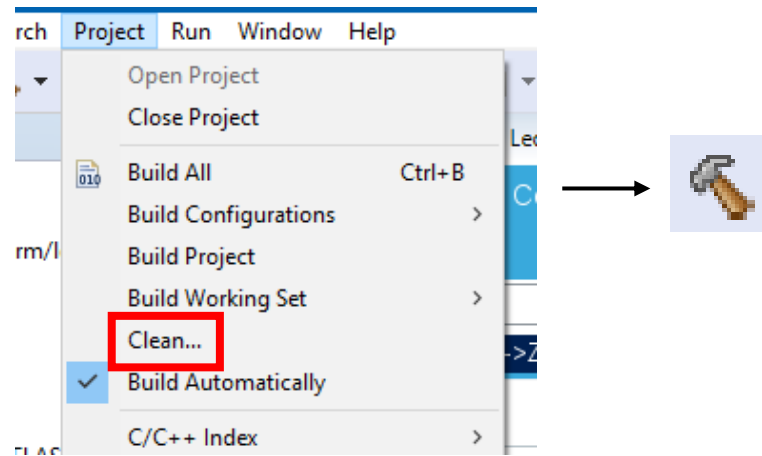
2. Always build first, inspect 'Console' window, then 'Debug' (if there are no errors)



Hints/ Wenke

3. Try to 'Clean' build if you get unexpected behaviour (Clean = remove all the existing object files and build meta-files).

- Clean followed by Build

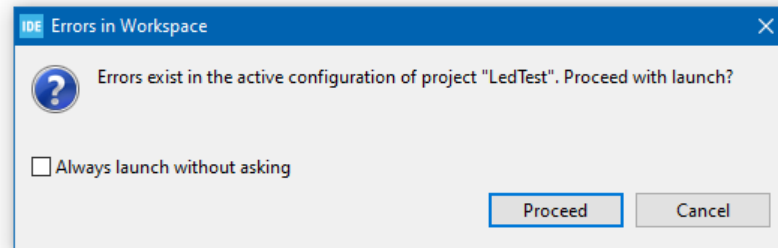


- Now, look at the console window to determine if there are problems

Common problems / Algemene probleme

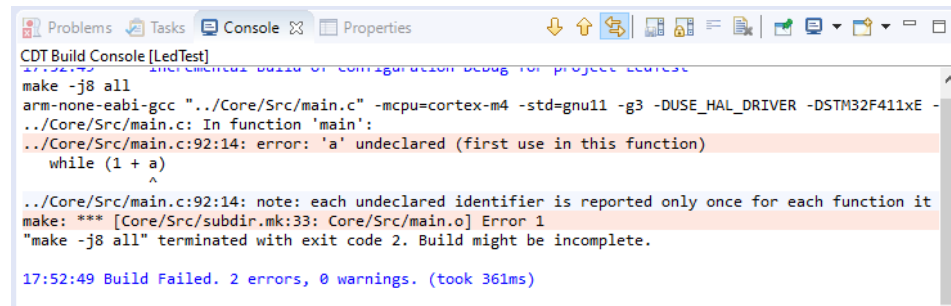
(after clicking Debug or Run  )

1. Errors exist in the active configuration of project...



Cause

There is an error in your code, causing the build process to fail. Look at the 'console' window for the reason.



```
CDT Build Console [LedTest]
17:52:49 Incremental build or configuration debug for project 'LedTest'
make -j8 all
arm-none-eabi-gcc "../Core/Src/main.c" -mcpu=cortex-m4 -std=gnu11 -g3 -DUSE_HAL_DRIVER -DSTM32F411xE -
../Core/Src/main.c: In function 'main':
../Core/Src/main.c:92:14: error: 'a' undeclared (first use in this function)
    while (1 + a)
               ^
../Core/Src/main.c:92:14: note: each undeclared identifier is reported only once for each function it
make: *** [Core/Src/subdir.mk:33: Core/Src/main.o] Error 1
"make -j8 all" terminated with exit code 2. Build might be incomplete.

17:52:49 Build Failed. 2 errors, 0 warnings. (took 361ms)
```

Solution

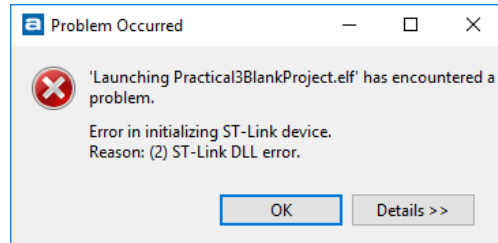
Fix the code error.

Note that if you press 'Proceed', the IDE will continue to program and run your previous binary file (assuming the build process was successful at some point)

Common problems / Algemene probleme

(after clicking Debug or Run  )

- 2. 'Launching ----' has encountered a problem.
Error initializing ST-Link device.



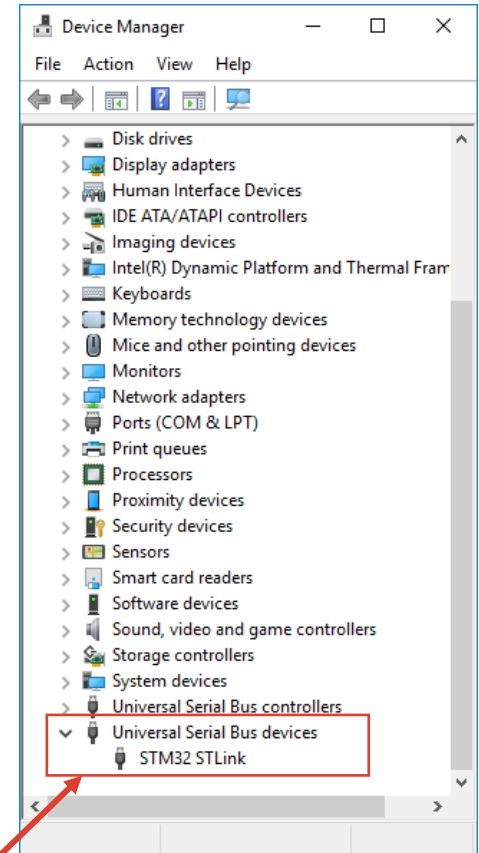
Cause

The IDE tried to program the binary file onto the development board, but could not. Possible causes

- The development board is not connected
- The cable or the development board is faulty
- The device driver for the STM board has not been installed

Solution

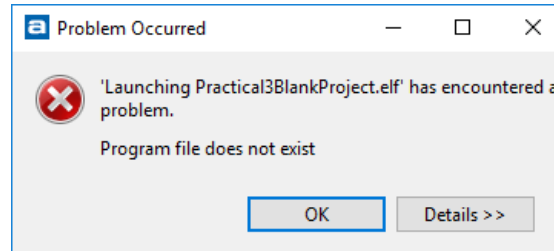
- Unplug and reconnect the development board.
- Open the Windows 'Device Manager'. If the board is plugged in, you should see an entry as in the image. If it is not there, there is a problem with the cable, development board or device driver.



Common problems / Algemene probleme

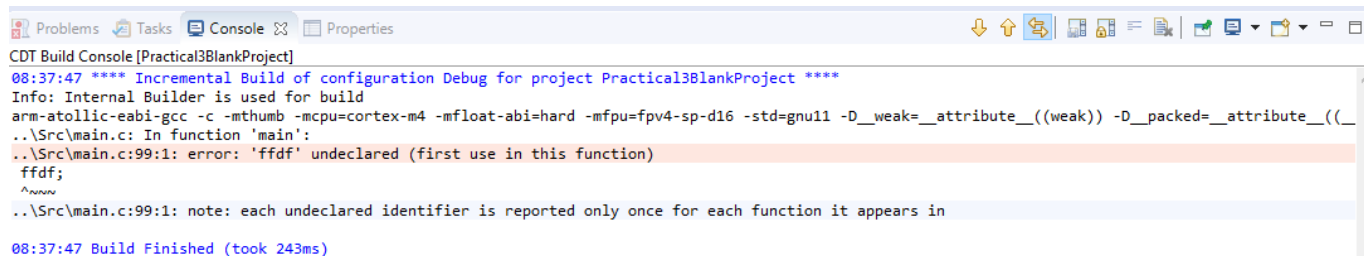
(after clicking Debug or Run  )

3. 'Launching ----' has encountered a problem. Program file does not exist



Cause

There is an error in your code, causing the build process to fail. Look at the 'console' window for the reason



```
CDT Build Console [Practical3BlankProject]
08:37:47 **** Incremental Build of configuration Debug for project Practical3BlankProject ****
Info: Internal Builder is used for build
arm-atollic-eabi-gcc -c -mthumb -mcpu=cortex-m4 -mfloat-abi=hard -mfpu=fpv4-sp-d16 -std=gnu11 -D__weak=__attribute__((weak)) -D__packed=__attribute__((__packed__)) -o ..\Src\main.o ..\Src\main.c: In function 'main':
..\Src\main.c:99:1: error: 'ffdf' undeclared (first use in this function)
    ffdf;
    ^~~~
..\Src\main.c:99:1: note: each undeclared identifier is reported only once for each function it appears in
08:37:47 Build Finished (took 243ms)
```

Solution

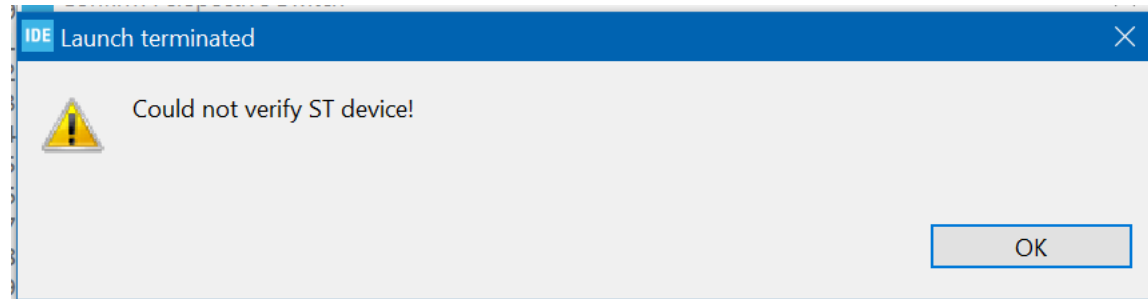
Fix the code error.

Common problems / Algemene problemen

(after clicking Debug or Run  )

4. Could not verify ST device!

- When using the Emulator



- You might be using STM32CubeIDE version 1.4. Please download version 1.3 from SunLearn and use that instead.

Mixing Assembly and C / Vermenging van Saamsteltaal en C kode

There are two ways to add ARM assembly code to your project:

1. As a separate assembly (.s) source file
2. Inline assembly using `asm()`

Mixing Assembly and C / Vermenging van Saamsteltaal en C code

Separate asm (.s) file example

```
.globl  set_leds
        .p2align 2
.type   set_leds,%function
.syntax unified
set_leds:                // Function "set_leds" entry point.
.fnstart
    push    {lr}
    lsl     r0, r0, #12  // Function argument in R0
    ldr     r3, =0x40020c14
    str     r0, [r3]
    pop     {pc}
.fnend
```



Mixing Assembly and C / Vermenging van Saamsteltaal en C code

Inline assembly example using asm()

```
void set_leds(uint32_t ledstate)
{
    asm(
        ".equ led_register,0x40020c14\n\t"
        "lsl    r0, r0, #12\n\t"
        "ldr     r3, =led_register\n\t"
        "str     r0, [r3]\n\t"
    );
}
```

or

```
void set_leds(uint32_t ledstate)
{
    asm(".equ led_register,0x40020c14" );
    asm(" lsl r0, r0, #12");
    asm(" ldr r3, =led_register");
    asm(" str r0, [r3]");
}
```

NB! Space or Tab (`\t`) is required
for non-labels (normal instructions)

Global variables and functions across source files/ Globale veranderlikes en funksies oor verskillende lêers

Often with global variables, you might want to access the variables in separate C source files.

In main.c

```
uint32_t count = 0; // global var

int main(void)
{
    while (1)
    {
        printf("%d\n", count);
    }
    return 0;
}
```

In other.c

```
void update(void)
{
    if (some_check)
    {
        count++; // compile error!
    }
}
```

>> 'count' undeclared (first use in this function)



Global variables and functions across source files/ Globale veranderlikes en funksies oor verskillende lêers

Often with global variables, you might want to access the variables in separate C source files.

Solution: tell the compiler the variable is declared elsewhere using the 'extern' keyword

In main.c

```
uint32_t count = 0; // global var

int main(void)
{
    while (1)
    {
        printf("%d\n", count);
    }
    return 0;
}
```

In other.c

```
// tell the compiler the global var
// is declared elsewhere
extern uint32_t count;

void update(void)
{
    if (some_check)
    {
        count++; // ok!
    }
}
```



Global variables and functions across source files/ Globale veranderlikes en funksies oor verskillende lêers

Often with global variables, you might want to access the variables in separate C source files.

Solution: making use of header files – (improved modularity)

In main.c

```
#include "other.h"

int main(void)
{
    while (1)
    {
        printf("%d\n", count);
    }
    return 0;
}
```

In other.c

```
// global var (this is the actual
// declaration)
uint32_t count = 0;

void update(void)
{
    if (some_check)
    {
        count++; // ok!
    }
}
```

In other.h

```
extern uint32_t count;
```



Global variables and functions across source files/ Globale veranderlikes en funksies oor verskillende lêers

Expanding on the previous idea, assume you have a software module, called "other" which has its own global variables and functions

In main.c

```
#include "other.h"

int main(void)
{
    while (1)
    {
        other_update();
        printf("%d\n", other_count);
    }
    return 0;
}
```

In other.c

```
// global var (this is the actual
// declaration)
uint32_t other_count = 0;

void other_update(void)
{
    if (some_check)
    {
        other_count++;
    }
}
```

In other.h

```
extern uint32_t other_count; // exported global variable
void other_update(void); // exported function prototype
```



Global variables and functions across source files/ Globale veranderlikes en funksies oor verskillende lêers

Remember

- Never `#include` a `.c` file (only `.h` files)
- Always use 'extern' for variables listed in the `.h` file
 - the implication is that variables are never defined in the header file, they are only made visible, or exported, from the `.h` file

Other

Please post on the SunLearn forum, if you encounter other issues that can be used to expand this file

