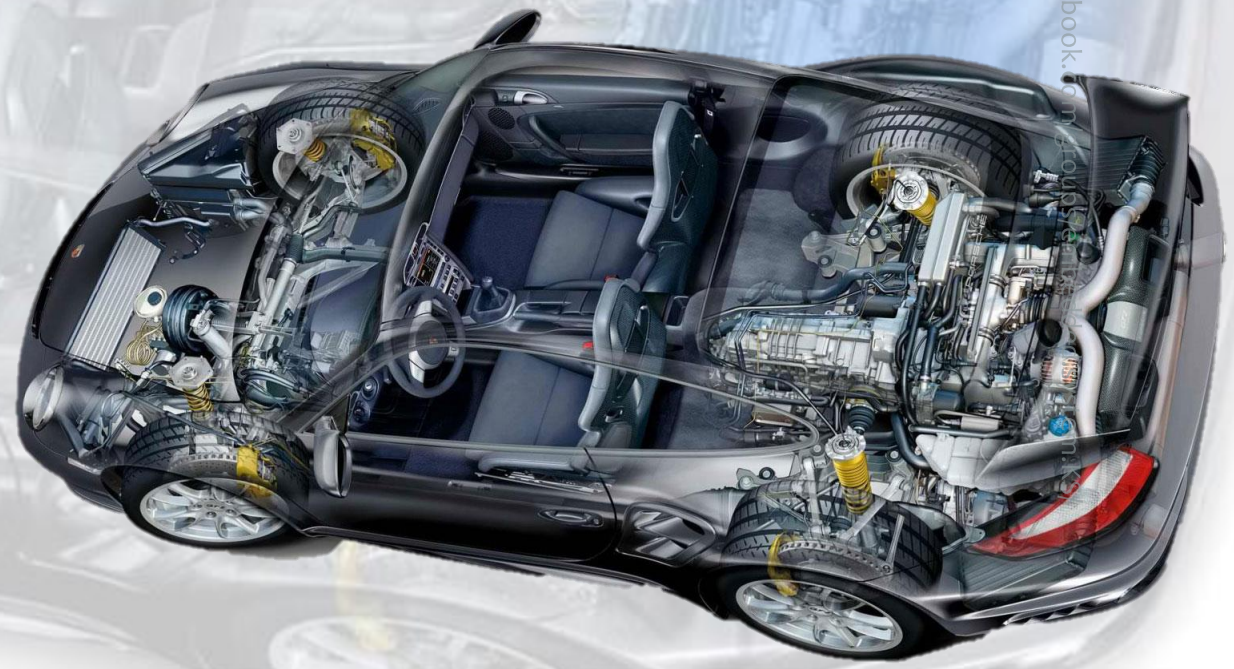
#LEARN_IN DEPTH

#Be_professional_in_embedded_system

Embedded C (unit3.Lesson1)

- TYPEDEF COMMAND
- HEADER PROTECTION
- OPTIMIZATION
- VOLATILE TYPE QUALIFIER
- CROSS-COMPILING TOOLCHAINS
- STATIC & DYNAMIC LIB
- COMPILATION PROCESS

ENG.KEROLES SHENOUDA

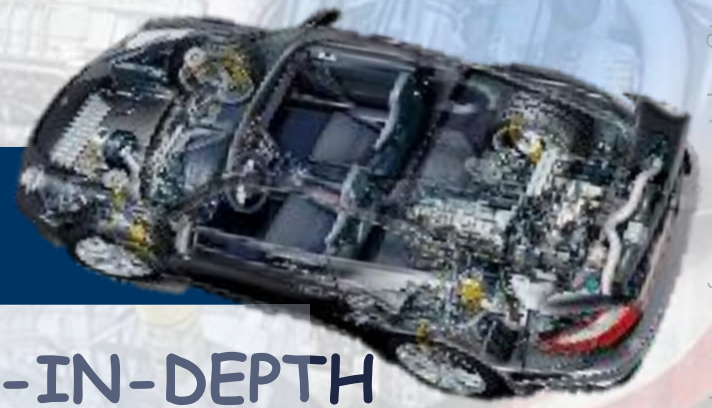


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

#LEARN_IN DEPTH

#Be_professional_in embedded_system

typedef command



LEARN-IN-DEPTH

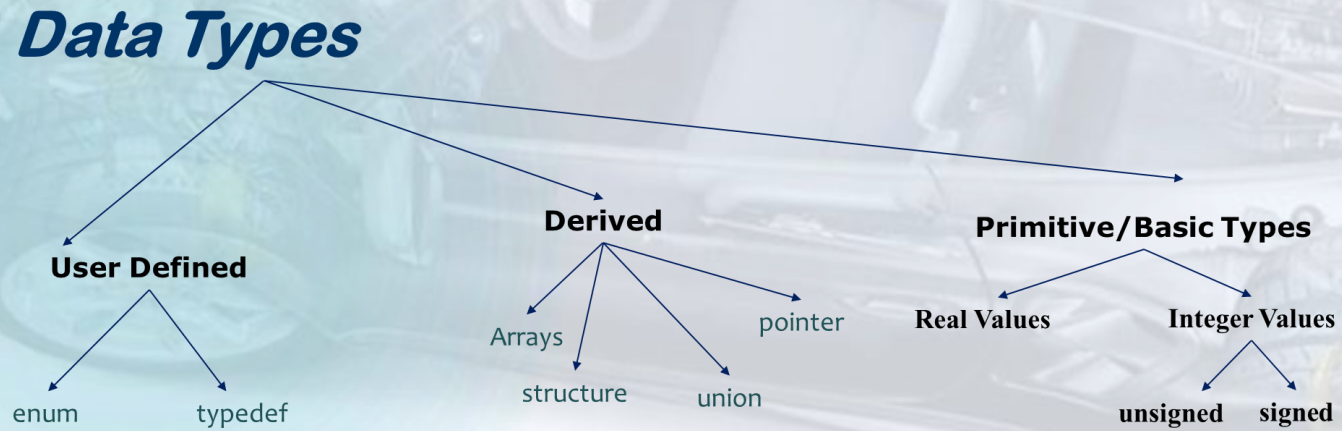
Be professional in

embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

typedef command

- **typedef** keywords in C is used to create an alias name to user defined data type to **primitive** and **Derived** datatype.
- `typedef <primitive or derived data type> <new_name> ;`



Note: Not mandatory just For readability and professional code :

1. Make NEW_NAME capital latter or like that New_Name
2. Put _t after MAME_t
3. Put S capital if struct and U if union at the beginning Sname_t
   If it is a Enum put n name_e

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

typedef command Cont.

```c
/*
 *  Created on: Oct 15, 2020
 *      Author: Keroles Shenouda
 */

#include "stdio.h"
typedef unsigned int Uint32_t ;
struct Sperson {
    Uint32_t weight ;
};
typedef struct Sperson   Sperson_t ;

int main ()
{
    Uint32_t x ;
    Sperson_t  y ;

    return 0 ;
}
```

Or

==

pointers.c

```c
/*
 *  Created on: Oct 15, 2020
 *      Author: Keroles Shenouda
 */

#include "stdio.h"
typedef unsigned int Uint32_t ;
typedef struct  {
    Uint32_t weight ;
} Sperson_t ;

int main ()
{
    Uint32_t x ;
    Sperson_t  y ;

    return 0 ;
}
```

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Let us look the data types in AUTOSAR Specification

AUTOSAR

Specification of Platform Types
V2.0.0

Document Title	Specification of Platform Types
Document Owner	AUTOSAR GbR
Document Responsibility	AUTOSAR GbR
Document Version	2.0.0
Document Status	Final

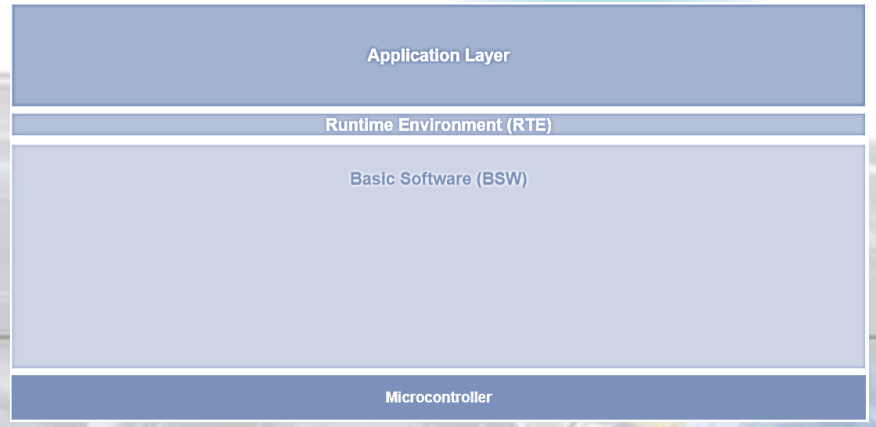
Document Change History

Date	Version	Changed by	Change Description
12.07.2006	2.0.0	AUTOSAR Administration	Second release
30.06.2005	1.0.0	AUTOSAR Administration	Initial Release

https://www.autosar.org/fileadmin/user_upload/standards/classic/2-0/AUTOSAR_SWS_PlatformTypes.pdf?fbclid=IwAR1N7-7H17vYZ8GAH5LofcVF0eMUwKi4PT3bdxAPG7_xP__W7rwvr6SX7qU

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

the data types in AUTOSAR Specification



5.1.2.1 Communication related basic software modules

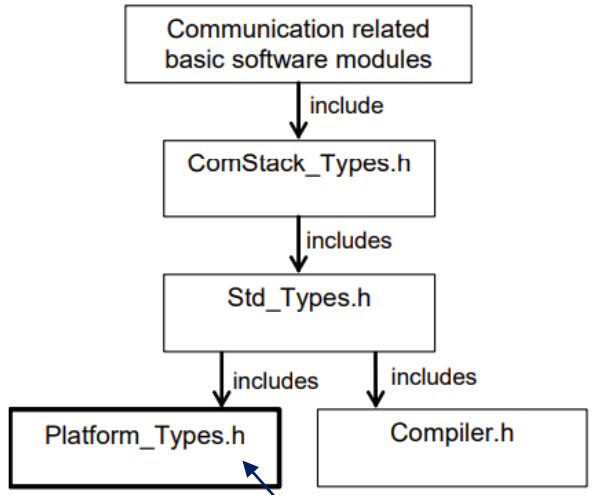


Figure 1: Include File Structure for communication related basic software modules

5.1.3 Non-communication related basic software modules

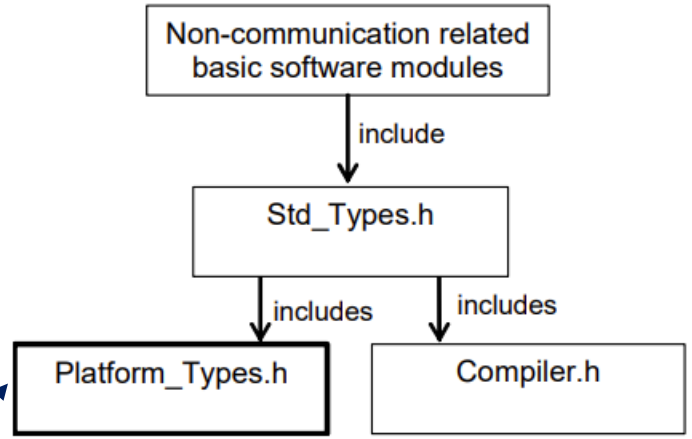


Figure 2: Include File Structure for non-communication related basic software modules

Focus on

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Type definition for different architecture in AUTOSAR

12.2 Type definitions – S12X

PLATFORM006: The platform types for Freescale S12X mapping to the ANSI C types:

Symbols:

```
#define CPU_TYPE            CPU_TYPE_16
#define CPU_BIT_ORDER       LSB_FIRST
#define CPU_BYTE_ORDER      HIGH_BYTE_FIRST
```

Types:

```
typedef unsigned char       boolean;

typedef signed char         sint8;
typedef unsigned char       uint8;
typedef signed short        sint16;
typedef unsigned short      uint16;
typedef signed long         sint32;
typedef unsigned long       uint32;

typedef signed char         sint8_least;
typedef unsigned char       uint8_least;
typedef signed short        sint16_least;
typedef unsigned short      uint16_least;
typedef signed long         sint32_least;
typedef unsigned long       uint32_least;

typedef float               float32;
typedef double              float64;
```

12.4 Type definitions – ST30

PLATFORM008: The platform types for STMicroelectronics ST30 following mapping to the ANSI C types:

Symbols:

```
#define CPU_TYPE            CPU_TYPE_32
#define CPU_BIT_ORDER       LSB_FIRST
#define CPU_BYTE_ORDER      LOW_BYTE_FIRST
```

Types:

```
typedef unsigned long       boolean;

typedef signed char         sint8;
typedef unsigned char       uint8;
typedef signed short        sint16;
typedef unsigned short      uint16;
typedef signed long         sint32;
typedef unsigned long       uint32;
```

12.10 Type definitions – SHx

PLATFORM059: The platform types for Renesas SHx mapping to the ANSI C types:

Symbols:

```
#define CPU_TYPE            CPU_TYPE_32
#define CPU_BIT_ORDER       LSB_FIRST
#define CPU_BYTE_ORDER      HIGH_BYTE_FIRST
```

Types:

```
typedef unsigned int        bool;

typedef signed char         sint8;
typedef unsigned char       uint8;
typedef signed short        sint16;
typedef unsigned short      uint16;
typedef signed int          sint32;
typedef unsigned int        uint32;
```

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

Also you can use #include "stdint.h"

pointers.c | stdint.h

```
26 /* 7.18.1.1  Exact-width integer types */
27 typedef signed char int8_t;
28 typedef unsigned char   uint8_t;
29 typedef short  int16_t;
30 typedef unsigned short  uint16_t;
31 typedef int  int32_t;
32 typedef unsigned   uint32_t;
33 typedef long long  int64_t;
34 typedef unsigned long long   uint64_t;
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Platform_Types.h

pointers.c | stdint.h | stdbool.h | platform_Types.h

```c
/*
 *  Created on: Oct 15, 2020
 *      Author: Keroles Shenouda
 *      Platform_Types.h
 */

#ifndef PLATFORM_TYPES_H_
#define PLATFORM_TYPES_H_


#include <stdbool.h>
#include <stdint.h>


#ifndef _Bool
#define _Bool unsigned char
#endif

#define CPU_TYPE            CPU_TYPE_32
#define CPU_BIT_ORDER       MSB_FIRST
#define CPU_BYTE_ORDER      HIGH_BYTE_FIRST

#ifndef FALSE
#define FALSE       (boolean)false
#endif
#ifndef TRUE
#define TRUE        (boolean)true
#endif
```

```c
typedef _Bool                   boolean;
typedef int8_t                  sint8;
typedef uint8_t                 uint8;
typedef char                    char_t;
typedef int16_t                 sint16;
typedef uint16_t                uint16;
typedef int32_t                 sint32;
typedef uint32_t                uint32;
typedef int64_t                 sint64;
typedef uint64_t                uint64;

typedef volatile int8_t vint8_t;
typedef volatile uint8_t vuint8_t;

typedef volatile int16_t vint16_t;
typedef volatile uint16_t vuint16_t;

typedef volatile int32_t vint32_t;
typedef volatile uint32_t vuint32_t;

typedef volatile int64_t vint64_t;
typedef volatile uint64_t vuint64_t;

#endif /* PLATFORM_TYPES_H_ */
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

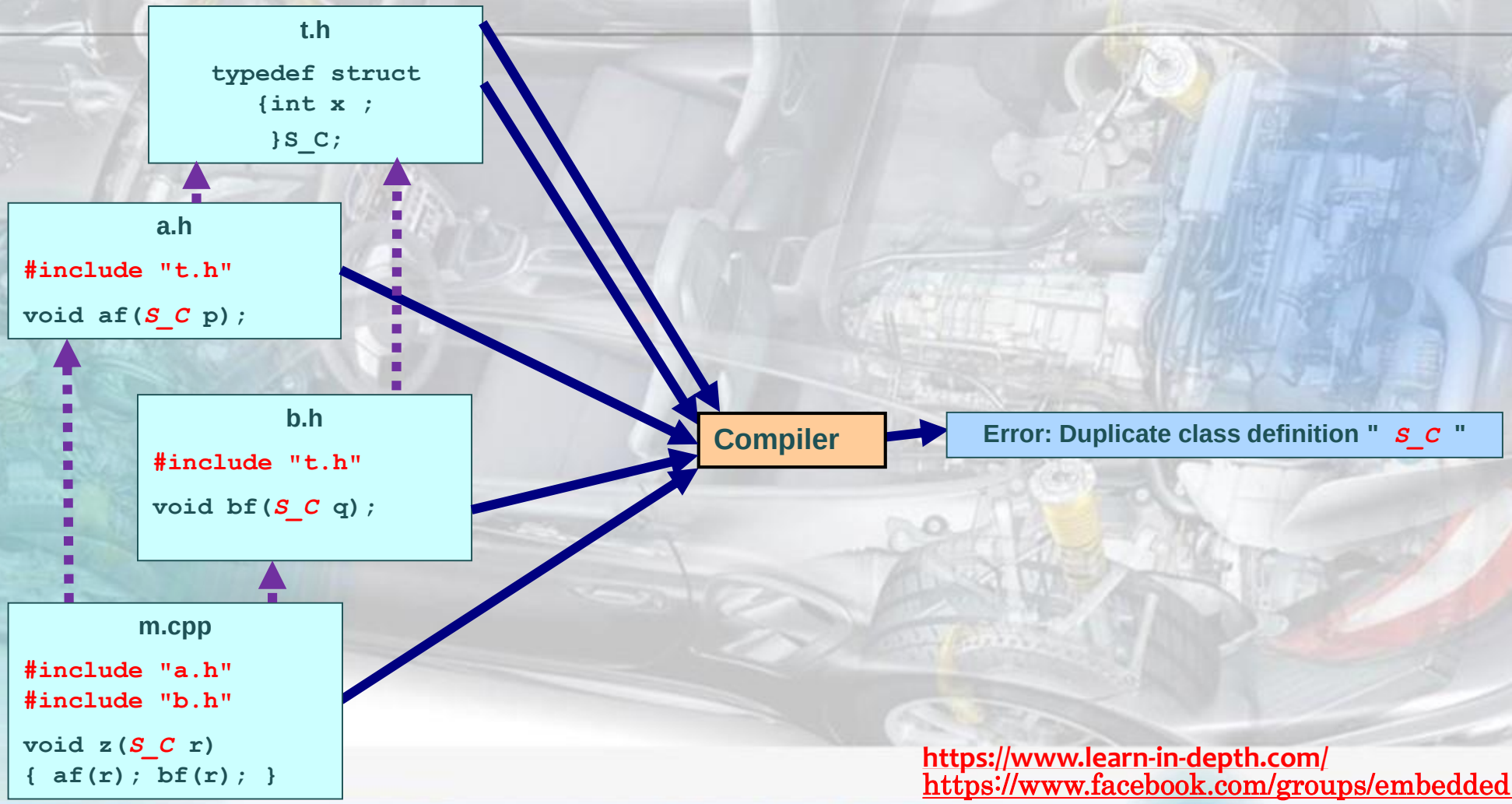
Header protection

LEARN-IN-DEPTH.COM ;)

LEARN-IN-DEPTH
Be professional in
embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

violation



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Header Protection

t.hpp

```
 8 #ifndef T_H_
 9 #define T_H_
10
11 typedef struct   {
12     int x ;
13 } S_C ;
14
15 #endif /* T_H_ */
16
```

a.hpp

```
 3 #ifndef A_H_
 4 #define A_H_
 5 #include "t.h"
 6 void af(S_C q);
 7 #endif
 8
```

b.hpp

```
 8 #ifndef B_H_
 9 #define B_H_
10 #include "t.h"
11 void bf(S_C p);
12 #endif /* B_H_ */
13
```

m.cpp

```
#include "a.hpp"
#include "b.hpp"

void z(C r)
{ af(r); bf(r); }
```

```
 9 #include "stdio.h"
10 #include "a.h"
11 #include "b.h"
12
13 int main ()
14 {
15     S_C r ;
16     af(r);
17     bf(r);
18     return 0 ;
19 }
20 void bf(S_C q)
21 {
22     printf ("bf");
23 }
24 void af(S_C p)
25 {
26     printf ("af");
27
28 }
```

Compiler → OK

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

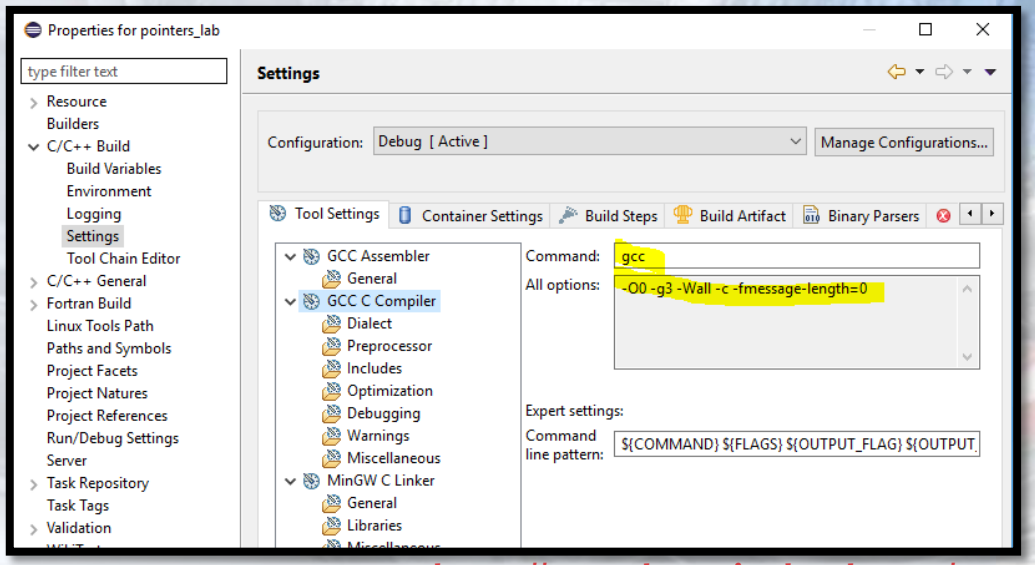
Optimization

LEARN-IN-DEPTH
Be professional in
embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Optimization

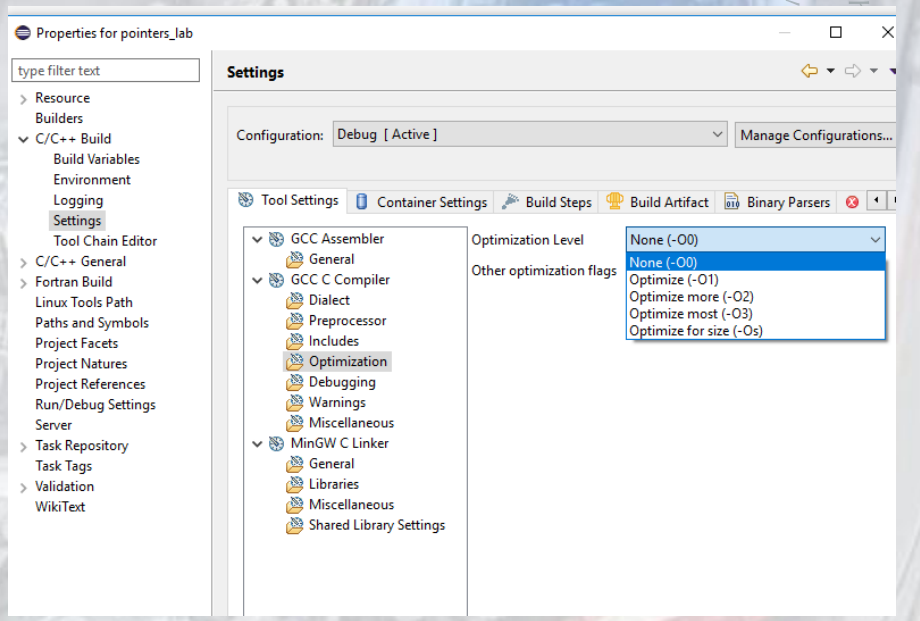
- The Optimization is a series of actions taken by the compiler on your program's code generation process to reduce:
  - number of instructions (code space optimization
  - Memory access time (time space optimization)
  - Power consumption
- Optimization level
  - Optimization level O0
  - Optimization level O1
  - Optimization level O2
  - Optimization level O3



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Optimization level O0

- No optimization
- Not recommmeneded for productions if you have limited code and ram space
- Has fastest compilation time
- This is debugging friendly and used during development
- A code which work with O0 may **not work with O0+ optimization levels**.
  - Code needs to be verified again applying higher optimization levels.



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

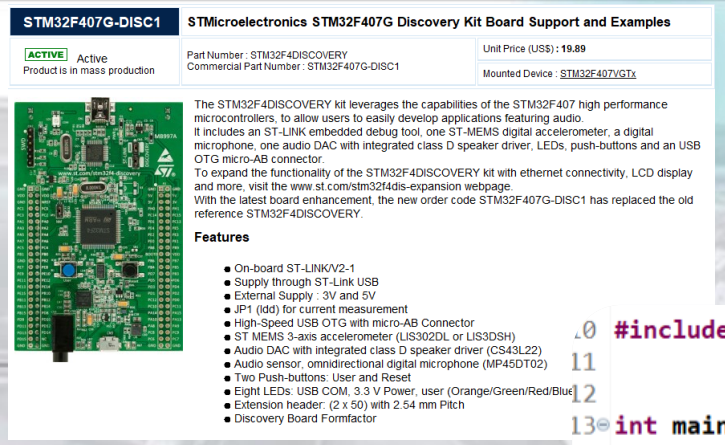
Optimization level O+

- Optimization level O1
  - Moderate optimization to decrease memory access and code space
- Optimization level O2
  - Full optimization
  - Slow compilation time
  - Not debugging friendly
- Optimization level O3
  - Full optimization of O2 + some more aggressive steps will be taken by the compiler.
  - Slowest compilation time
  - May cause bugs in the program
  - Not debugging friendly

if the code crashes and the program doesn't end or gives unexpected results. In order to understand where is the problem it's necessary to open the Disassembly section in the Debugging mode and control in the register windows how the data is trasferred to registers

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Compile with –O3



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

```c
int main(void)
{
    uint8_t   data1;
    uint8_t  data2;

    data1 = 50;

    data2 = data1;

    data2 = data1;

    /* Loop forever */
    for(;;);
}
```

Thin in depth
Why size is difference

```c
#include<stdint.h>


int main(void)
{
    volatile uint8_t   data1;
    volatile uint8_t  data2;

    data1 = 50;

    data2 = data1;

    data2 = data1;

    /* Loop forever */
    for(;;);
}
```

```
Problems  Tasks  Console
CDT Build Console [Optimization_test]
arm-none-eabi-size   Optimization_test.elf
   text    data     bss     dec     hex filename
    692       8    1568    2268     8dc Optimization_test.elf
Finished building: default.size.stdout
Finished building: Optimization_test.list
Finished building: Optimization_test.hex

Finished building: Optimization_test.bin
```

```
Problems  Tasks
CDT Build Console [Optimization_test]
arm-none-eabi-size   Optimization_test.elf
arm-none-eabi-objcopy  -O ihex  Optimization_test.elf  "Optimization_test.hex"
   text    data     bss     dec     hex filename
    672       8    1568    2248     8c8 Optimization_test.elf
Finished building: default.size.stdout

Finished building: Optimization_test.hex


19:54:21 Build Finished. 0 errors, 0 warnings. (took 357ms)
```

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

With –O3

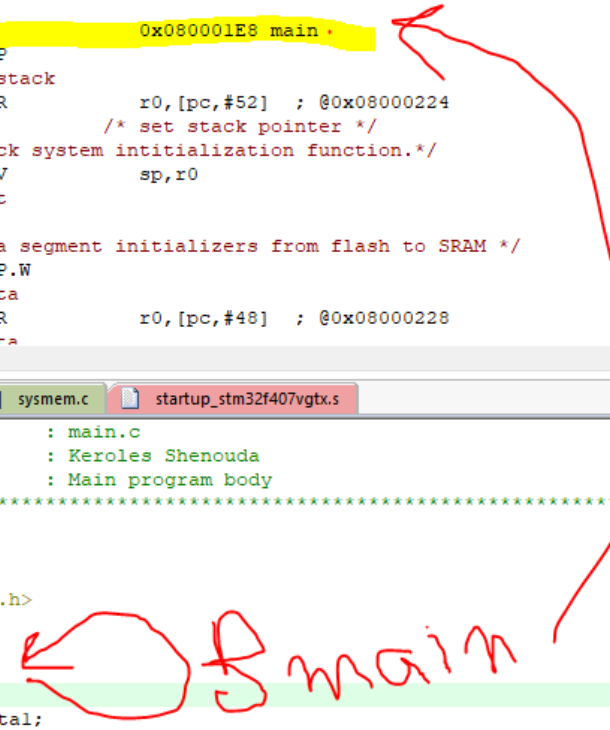
```
Problems  Tasks  Console  Properties
CDT Build Console [Optimization_test]
arm-none-eabi-size   Optimization_test.elf
arm-none-eabi-objcopy  -O ihex  Optimization_test.elf  "Optimization_test.hex"
   text    data     bss     dec     hex filename
    672       8    1568    2248     8c8 Optimization_test.elf
Finished building: default.size.stdout

Finished building: Optimization_test.hex


19:54:21 Build Finished. 0 errors, 0 warnings. (took 357ms)
```

Disassembly

```
0x080001DE 0000      MOVS        r0,r0
0x080001E0 0004      MOVS        r4,r0
0x080001E2 2000      MOVS        r0,#0x00
0x080001E4 0288      LSLS        r0,r1,#10
0x080001E6 0800      LSRS        r0,r0,#0
    14: {
0x080001E8 E7FE      B           0x080001E8 main
0x080001EA BF00      NOP
    59:   ldr   r0, =_estack
0x080001EC 480D      LDR         r0,[pc,#52]  ; @0x08000224
    60:   mov   sp, r0          /* set stack pointer */
    61: /* Call the clock system intitialization function.*/
0x080001EE 4685      MOV         sp,r0
    62:   bl  SystemInit
    63:
    64: /* Copy the data segment initializers from flash to SRAM */
0x080001F0 F3AF8000  NOP.W
    65:   ldr r0, =_sdata
0x080001F4 480C      LDR         r0,[pc,#48]  ; @0x08000228
```

main.c | syscalls.c | sysmem.c | startup_stm32f407vgtx.s

```
 3  * @file           : main.c
 4  * @author         : Keroles Shenouda
 5  * @brief          : Main program body
 6  ******************************************************************************
 7
 8  */
 9
10  #include<stdint.h>
11
12
13  int main(void)
14  {
15     uint8_t   data1;
16     uint8_t  data2;
17
18    data1 = 50;
19
20    data2 = data1;
21
22    data2 = data1;
23
24      /* Loop forever */
25    for(;;);
26  }
27
```



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

With –O3

```
Disassembly
    14: {
    15:         volatile uint8_t   data1;
    16:         volatile uint8_t  data2;
    17:
0x080001E8 B082      SUB           sp,sp,#0x08
    18:         data1 = 50;
    19:
0x080001EA 2332      MOVS          r3,#0x32
0x080001EC F88D3006  STRB          r3,[sp,#0x06]
    20:         data2 = data1;
    21:
0x080001F0 F89D3006  LDRB          r3,[sp,#0x06]
0x080001F4 B2DB      UXTB          r3,r3
0x080001F6 F88D3007  STRB          r3,[sp,#0x07]
    22:         data2 = data1;
0x080001FA F89D3006  LDRB          r3,[sp,#0x06]
0x080001FE B2DB      UXTB          r3,r3
0x08000200 F88D3007  STRB          r3,[sp,#0x07]
0x08000204 E7FE      B             0x08000204
0x08000206 BF00      NOP
```

main.c | syscalls.c | sysmem.c | startup_stm32f407vgtx.s

```
 3  * @file           : main.c
 4  * @author         : Keroles Shenouda
 5  * @brief          : Main program body
 6  ******************************************************************
 7
 8  */
 9
10  #include<stdint.h>
11
12
13  int main(void)
14  {
15    volatile uint8_t   data1;
16    volatile uint8_t  data2;
17
18    data1 = 50;
19
20    data2 = data1;
21
22    data2 = data1;
23
24      /* Loop forever */
25    for(;;);
26  }
27
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

#LEARN_IN DEPTH

#Be_professional_in embedded_system

Volatile Type Qualifier

VERY IMPORTANT IN EMBEDDED C

LEARN-IN-DEPTH

Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Volatile Type Qualifier

28.Volatile Qualifier in C Program :

C's volatile keyword is a qualifier that is applied to a variable when it is declared.

It tells the compiler that the value of the variable may change at any time--without any action being taken by the code.

So it can't optimize in such a way that repeated references to the value of the variable don't access the actual variable's storage for each reference.

"Always Read From Memory" - No Optimization Possible

Syntax :

```
volatile int X;
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Proper Use of C's volatile Keyword

- Memory-mapped peripheral registers
- Global variables modified by an interrupt service routine
- Global variables accessed by multiple tasks within a multi-threaded application

Syntax of C's volatile Keyword

- declare a variable volatile, include the keyword volatile before or after the data type in the variable definition. For instance both of these declarations will declare foo to be a volatile integer:
  - volatile int foo;
  - int volatile foo;

volatile

- Now, it turns out that pointers to volatile variables are very common, especially with memory-mapped I/O registers. Both of these declarations declare pReg to be a pointer to a volatile unsigned 8-bit integer:
  - volatile uint8_t * pReg;
  - uint8_t volatile * pReg;

pointer to a volatile unsigned 8-bit integer

volatile

- Volatile pointers to non-volatile data
  - int * volatile p;

Volatile pointers to non-volatile data

volatile

- a volatile pointer to a volatile variable:
  - int volatile * volatile p;

volatile pointer to a volatile variable

Peripheral Registers

- Embedded systems contain real hardware, usually with sophisticated peripherals. These peripherals contain registers whose values may change asynchronously to the program flow. As a very simple example, consider an 8-bit **status** register that is memory mapped at address 0x1234. It is required that you poll the status register until it becomes non-zero.

incorrect implementation

```
uint8_t * pReg = (uint8_t *) 0x1234;
 // Wait for register to become non-
//zero while
(*pReg == 0) { } // Do something else
```

```
mov ptr, #0x1234
mov a, @ptr
loop:
 bz loop
```

assembly

This will almost certainly fail as soon as you turn compiler optimization on, since the compiler will generate assembly language that looks something like this:

Optimization cause issue

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

To force the compiler to do what we want, we modify the declaration to:

```
uint8_t volatile * pReg = (uint8_t volatile *) 0x1234;

//The assembly language now looks like this:

mov ptr, #0x1234
loop:
    mov a, @ptr
    bz loop
```

No Optimization here

Interrupt Service Routines

```
int etx_rcvd = FALSE;

void main()
{
   ...
   while (!ext_rcvd)
   {
      // Wait
   }
   ...
}

//Interrupt
 void rx_isr(void)
{
   ...
   if (ETX == rx_char)
   {
      etx_rcvd = TRUE;
   }
   ...
}
```

Optimization cause issue

The solution is to declare the variable etx_rcvd to be volatile

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Multithreaded Applications

```
int cntr;

void task1(void)
{
   cntr = 0;

   while (cntr == 0)
   {
      sleep(1);
   }
   ...
}

void task2(void)
{
   ...
   cntr++;
   sleep(10);
   ...
}
```

Optimization cause issue

This code will likely fail once the compiler's optimizer is enabled. Declaring cntr to be volatile is the proper way to solve the problem.

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Now lets see how access the register absolute address

- Write 0xFFFFFFFF on SIU register which have absolute address 0x30610000

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Solution 1

- Declare a pointer,
  - volatile int *p;
- Assign the address of the I/O memory location to the pointer,
  - p = (volatile int*) 0x30610000;
    /* 4-byte long, address of some memory-mapped register */
- Output a 32-bit value
  - *p = 0xFFFFFFFF;

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Solution 2

- *((volatile unsigned long*)(0x306100)) = 0xFFFFFFFF;
- Or
- #define MYREGISTER *((volatile unsigned long*)(0x306100))

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Solution 3

```
typedef union {
      vuint32_t   ALL_ports;
      struct {
            vuint32_t   PORTA:8 ;
            vuint32_t   PORTB:8 ;
            vuint32_t   PORTC:8 ;
            vuint32_t   PORTD:8 ;
            } SIU_fields;
 } SIU_R;
```


```
volatile   SIU_R*  PORTS =  (volatile   SIU_R* ) 0x306100 ;   /*Address*/
PORTS ->ALL_ports  =0xFFFFFFFF;  // access all BITS
PORTS ->SIU_fields.PORTA = 0xFF ;
```

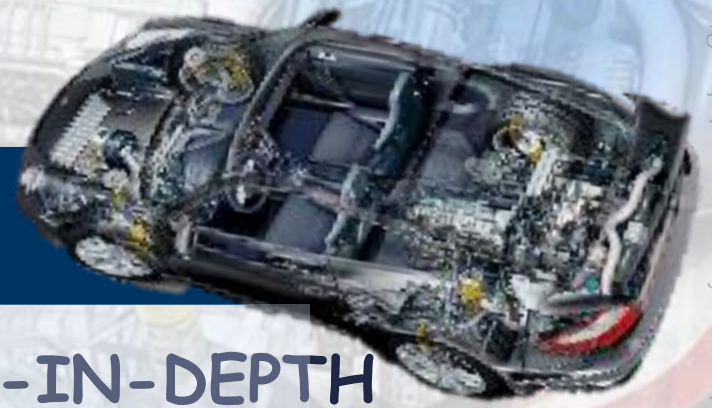

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

#LEARN_IN DEPTH

#Be_professional_in embedded_system

Lab _ live ☺ to understand well the volatile

THINK WITH ME ☺

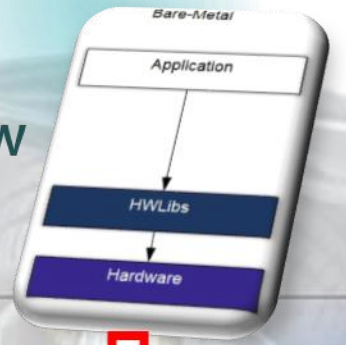


LEARN-IN-DEPTH

Be professional in
embedded system

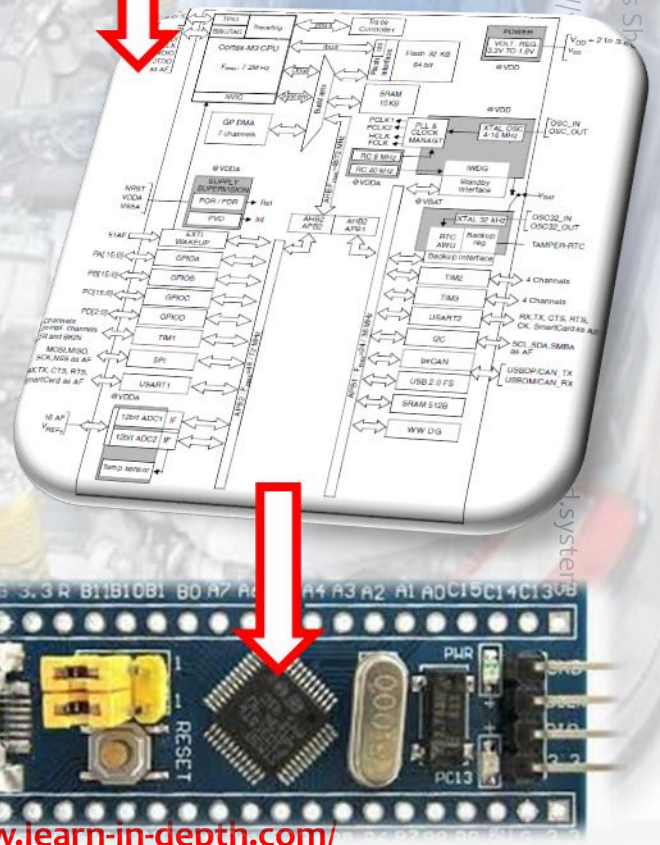
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Toole led on Stm32f103CX



Bare metal SW

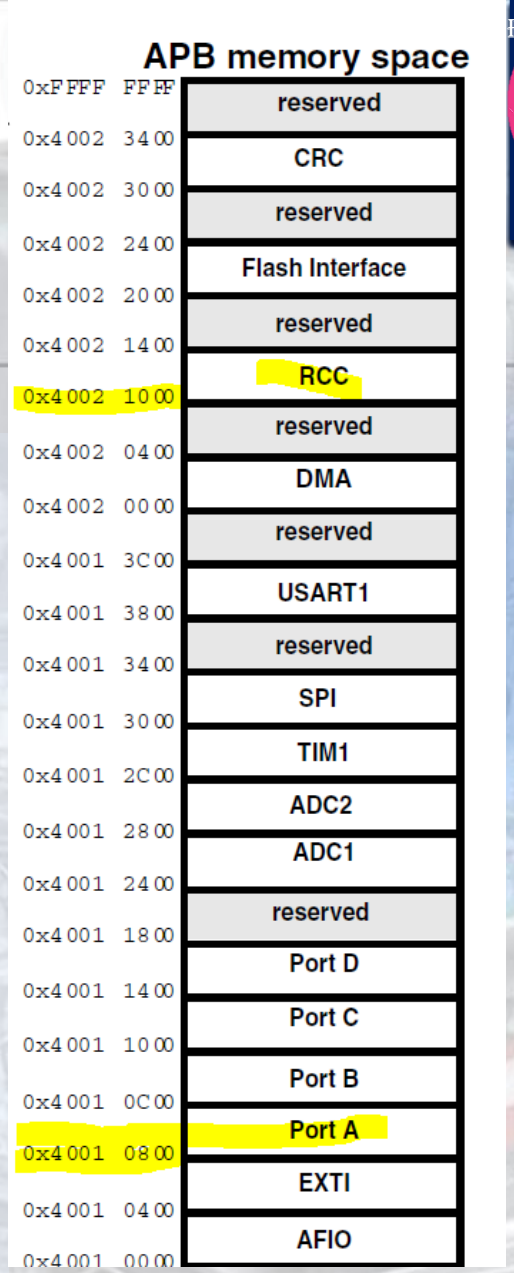
- Led is connected to GPIO port A13
- To make a GPIO toggling in STM32, you need to work with two peripherals:
  - RCC (reset and clock control)
    - The RCC is necessary because the GPIO has disabled clock by default.
  - GPIOx (general purpose input/output).



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

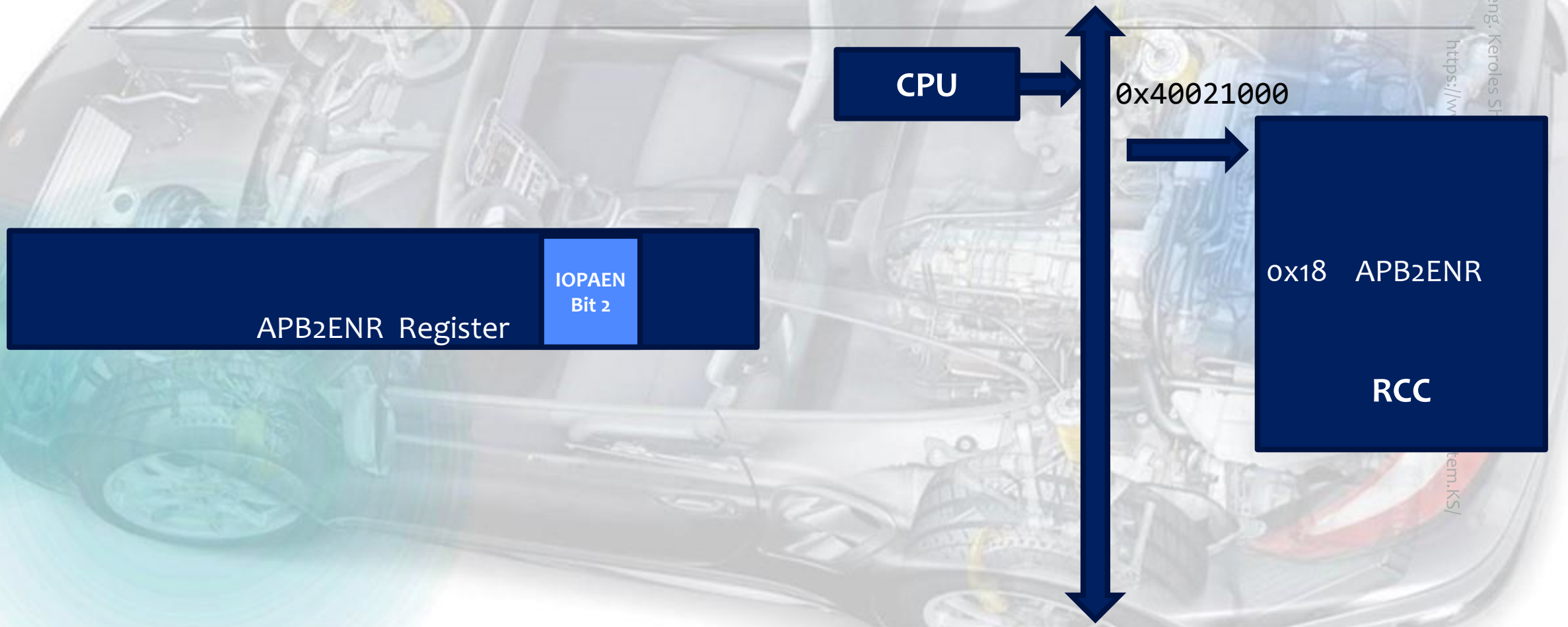
Toole led on Stm32f103CX

- Base address RCC
  - 0x40021000
- GPIO Port A
  - 0x40010800



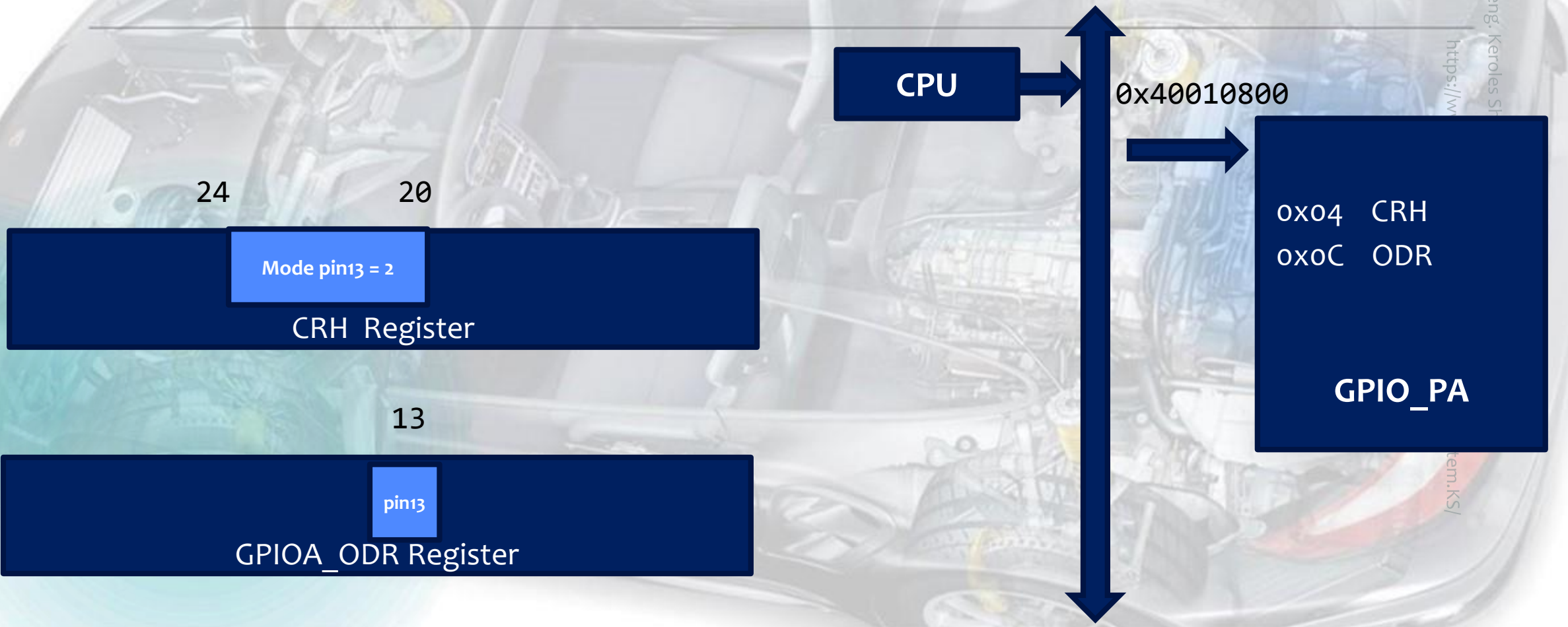
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Toole led on Stm32f103CX



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Toole led on Stm32f103CX

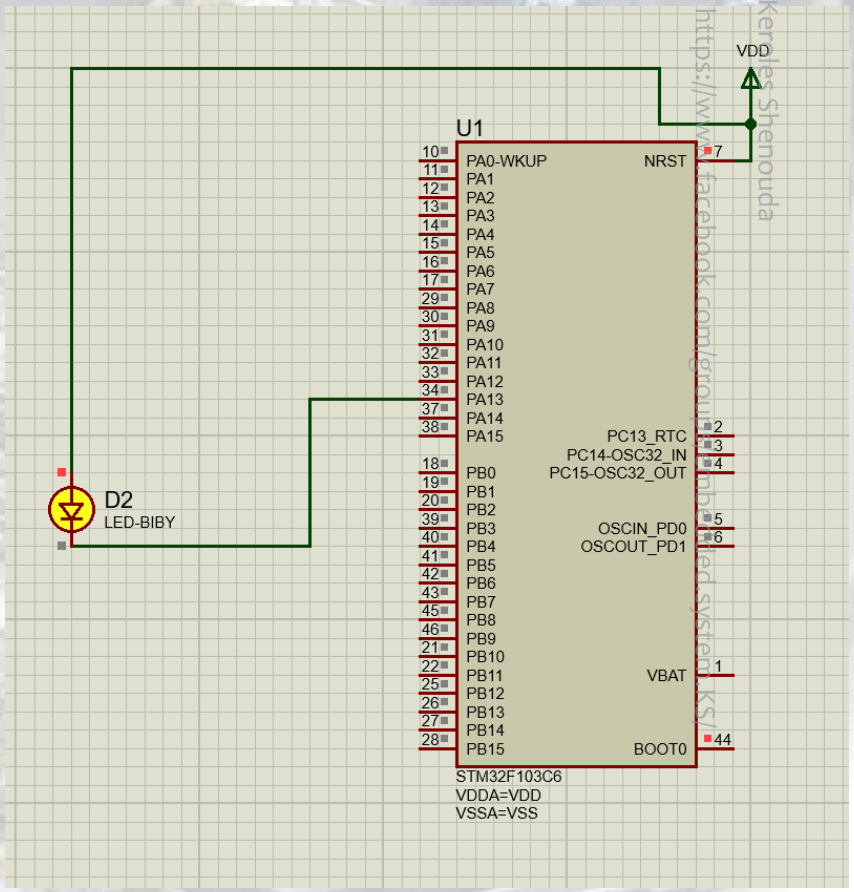


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

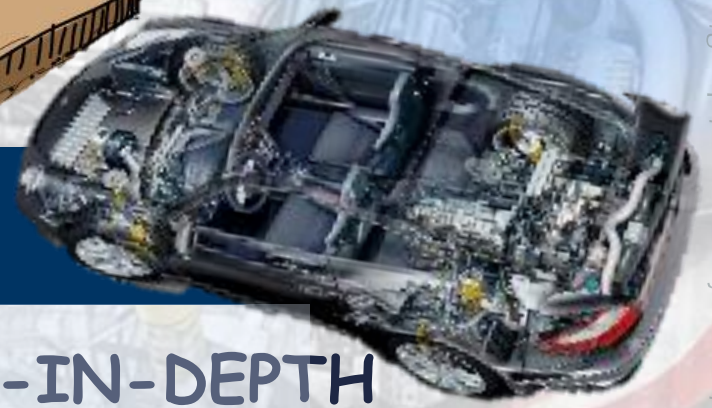
```c
/**
 ******************************************************************************
 * @file           : main.c
 * @author         : Keroles Shenouda
 * @brief          : Learn-in-depth
 ******************************************************************************
 */
typedef volatile unsigned int vuint32_t ;
#include <stdint.h>
// register address
#define RCC_BASE       0x40021000
#define GPIOA_BASE     0x40010800
#define RCC_APB2ENR    *(volatile uint32_t *)(RCC_BASE    + 0x18)
#define GPIOA_CRH      *(volatile uint32_t *)(GPIOA_BASE + 0x04)
#define GPIOA_ODR      *(volatile uint32_t *)(GPIOA_BASE + 0x0C)
// bit fields
#define RCC_IOPAEN    (1<<2)
#define GPIOA13       (1UL<<13)

typedef union  {
        vuint32_t     all_fields;
        struct {
        vuint32_t    reserved:13 ;
        vuint32_t    P_13:1 ;
        } Pin;
 } R_ODR_t;

volatile R_ODR_t*  R_ODR =   (volatile R_ODR_t*)(GPIOA_BASE + 0x0C) ;


int main(void)
{
    RCC_APB2ENR |= RCC_IOPAEN;
    GPIOA_CRH   &= 0xFF0FFFFF;
    GPIOA_CRH   |= 0x00200000;
    while(1)
    {
        R_ODR->Pin.P_13 =  1;
        for (int i = 0; i < 5000; i++); // arbitrary delay
        R_ODR->Pin.P_13 = 0;
        for (int i = 0; i < 5000; i++); // arbitrary delay
    }
}
```



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

#LEARN_IN DEPTH

#Be_professional_in embedded_system



Very important Question

LEARN-IN-DEPTH

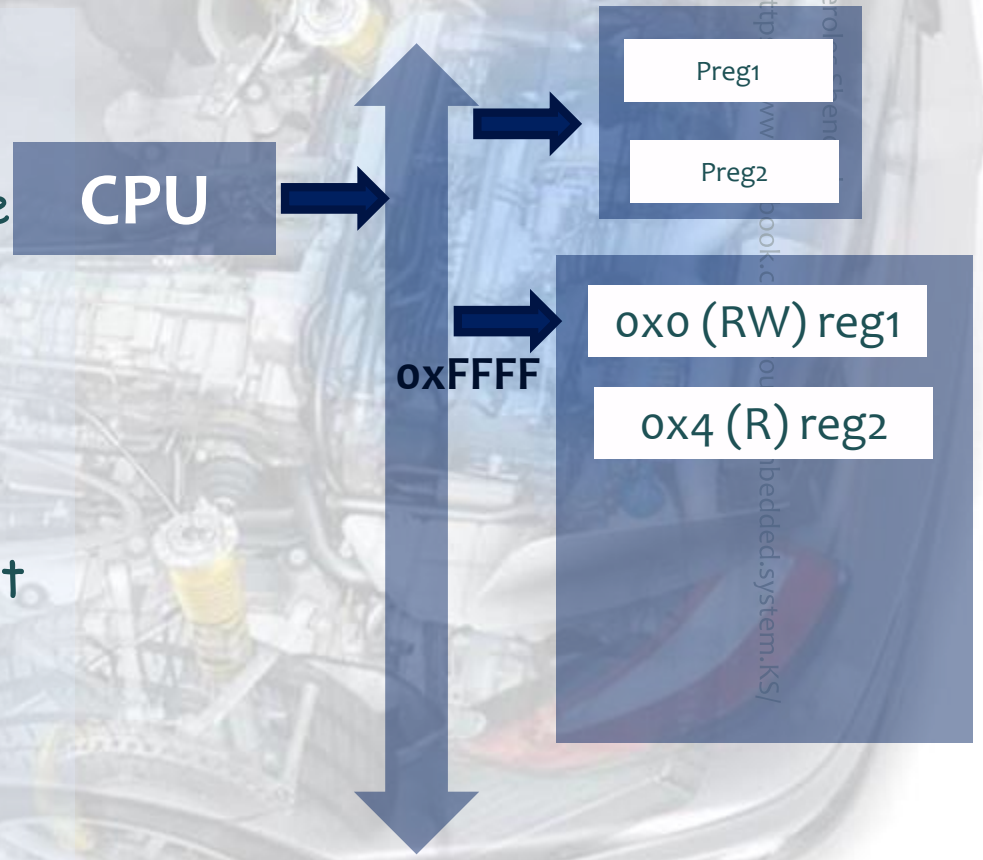
Be professional in

embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Usage of const and volatile together

- uint32_t volatile * const Preg1 = (uint32_t) 0xFFFF0000;
- **Preg1** is a const pointer pointing to volatile data of type uint32_t.
- **In this case const is used just to guard the pointer from unexpected changes from the programmer.**
- The volatile is used to tell compiler **that data pointed could receive unexpected changes**, the compiler has not to apply any optimization on that data then

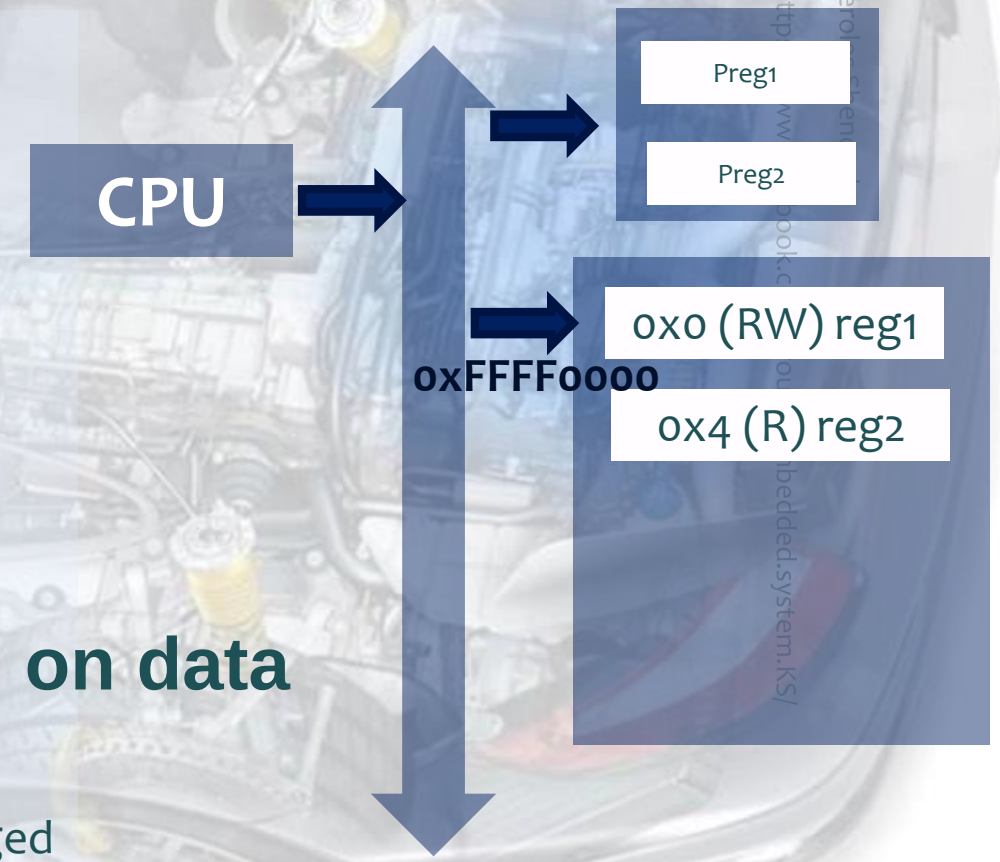


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Usage of const and volatile together

- `uint32_t const volatile * const Preg2 = (uint32_t) 0xFFFF0004;`
- **Preg2** is a **const** pointer pointing to **volatile**

const data of type uint32_t..



Interview Question
What that means ?
This qualifier (const+volatile) on data

It is fixed

It can be changed
At any time

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Usage of const and volatile together

- `uint32_t const volatile * const Preg2 = (uint32_t) 0xFFFF0004;`
- **Preg2** is a **const** pointer pointing to **volatile** **const** data of type uint32_t..

This qualifier (const+volatile) on data

means that data pointed by the indicated address is volatile (unexpected changes) but the **programmer shouldn't change** the data pointed to that address (const purpose).

In this case const is useful for (R read only access) register like the status register which is changed from the HW not from the programmer.

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

#LEARN_IN DEPTH

#Be_professional_in embedded_system

Bare metal Embedded SW

BE READY, THIS IS IMPORTANT PART

MOST OF THE INTERVIEW QUESTIONS IS CONCENTRATED HERE

LEARN-IN-DEPTH

Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

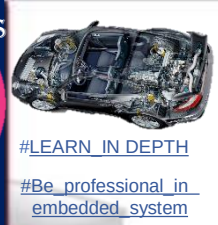
#LEARN_IN DEPTH

#Be_professional_in_embedded_system

How To Write Embedded C Code From Scratch without IDE ?

https://www.facebook.com/groups/embedded.system.KS/

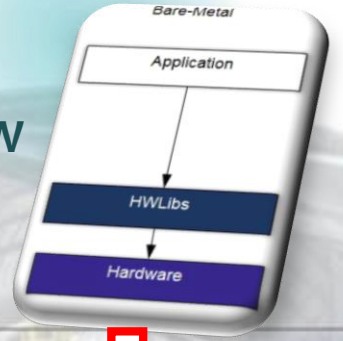
How To Write Embedded C Code From Scratch without IDE ?

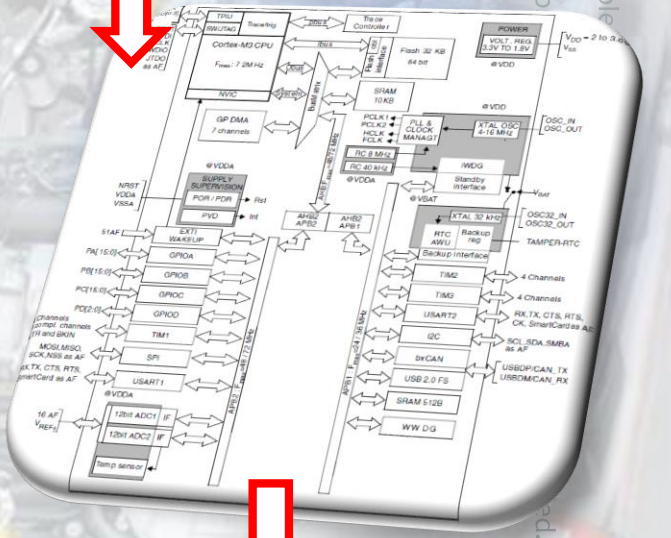


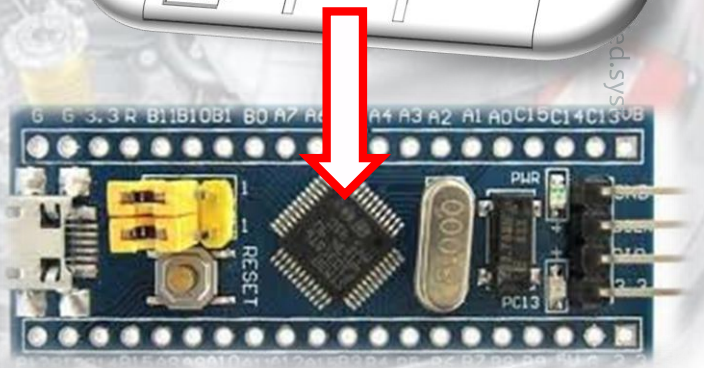
#LEARN_IN DEPTH

#Be_professional_in embedded_system

Bare metal SW







You will need

Startup.s

Linker Script

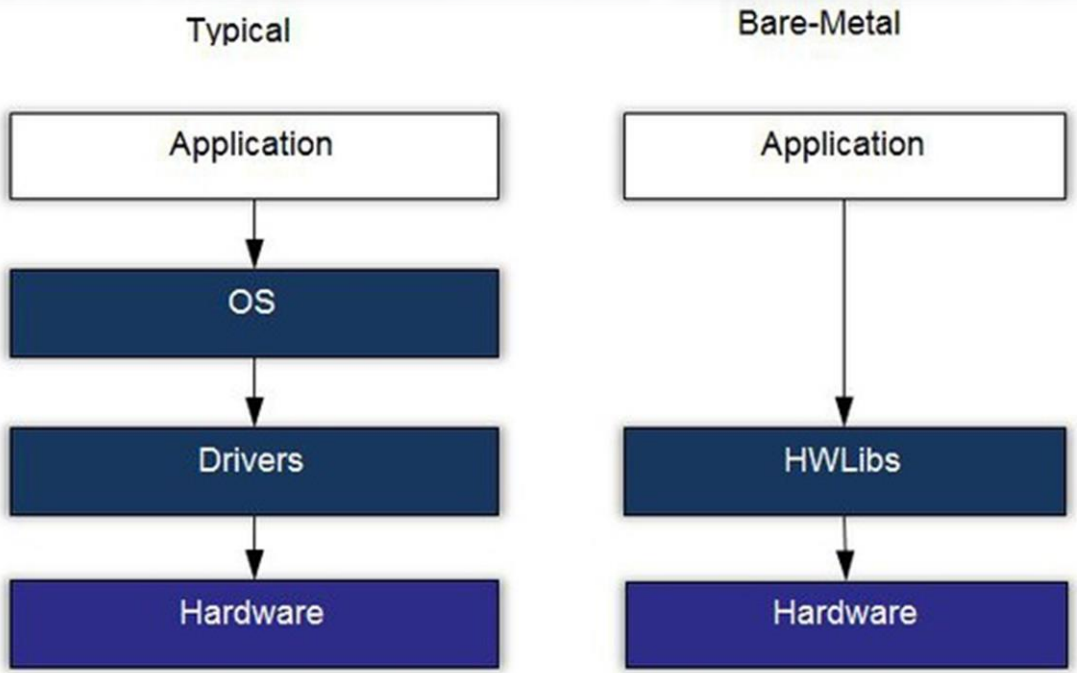
Cross Toolchain

Makefile

C Code files

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

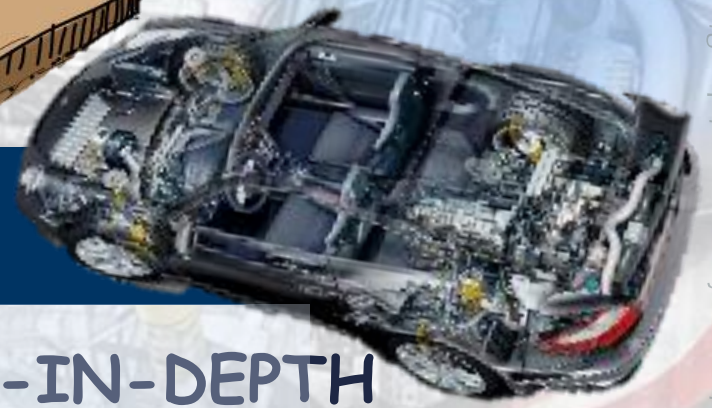
OS App Vs Bare metal SW



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

#LEARN_IN DEPTH

#Be_professional_in embedded_system



Cross-compiling toolchains

LEARN-IN-DEPTH

Be professional in

embedded system

https://www.learn-in-depth.com/

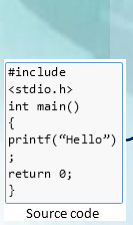
https://www.facebook.com/groups/embedded.system.KS/

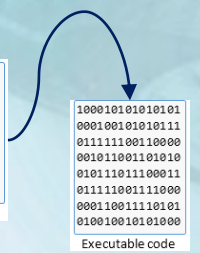
tool-chain

- The tool-chain consists of **a compiler**, **assembler**, **linker**, **debugger**, **libraries** and a number of helper programs.

For example
objcopy
objdump
readelf

- By default the **host Pc (Ubuntu, redhat, etc)** contains some development tools which ate called **as native toolchain**
- **Toolchain Components:**

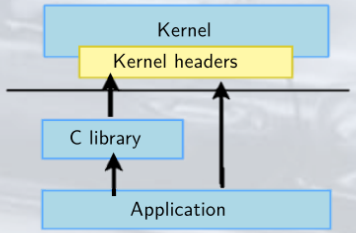




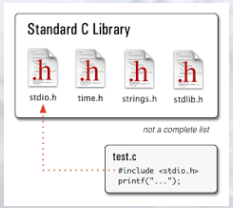
compiler

Binary Utilities

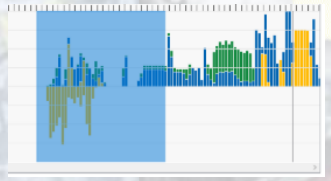
Debugger



Kernel Headers



C/C++ Libraries



Trace and profile (optional)

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

tool-chain Cont.

Instruction set:
Intel
OS: Linux/windows

C Project

Host machine
For Building process

Target machine
For run the executable

a.out

Instruction set:
AArch64
OS: Linux

C Project

Cross Compiling Toolchain

This toolchain runs run on **your workstation** but generate code for your **target**

Native Toolchain

This toolchain runs on **your workstation** and **generates code for your workstation,** usually x86

Instruction set:
Intel
OS: Linux/windows

C Project

Host machine
For Building process

Target machine
For run the executable

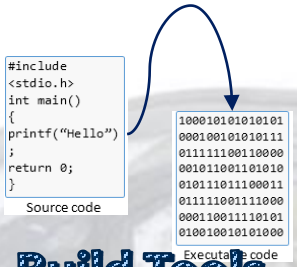
a.out

Instruction set:
Intel
OS: Linux/windows

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

#LEARN_IN DEPTH

#Be_professional_in embedded_system



Build Tools

Binary Utilities

Debugger

gcc	Compiling C/C++ (and other languages) code into object files Also it **calls a linker internally** to link object files into an executable
ld	**Links** object files into an executable (called internally by gcc)
make	Reads the **Makefile** to manage the **building process**
ar	**Archives** multiple Object files into **a static Library**

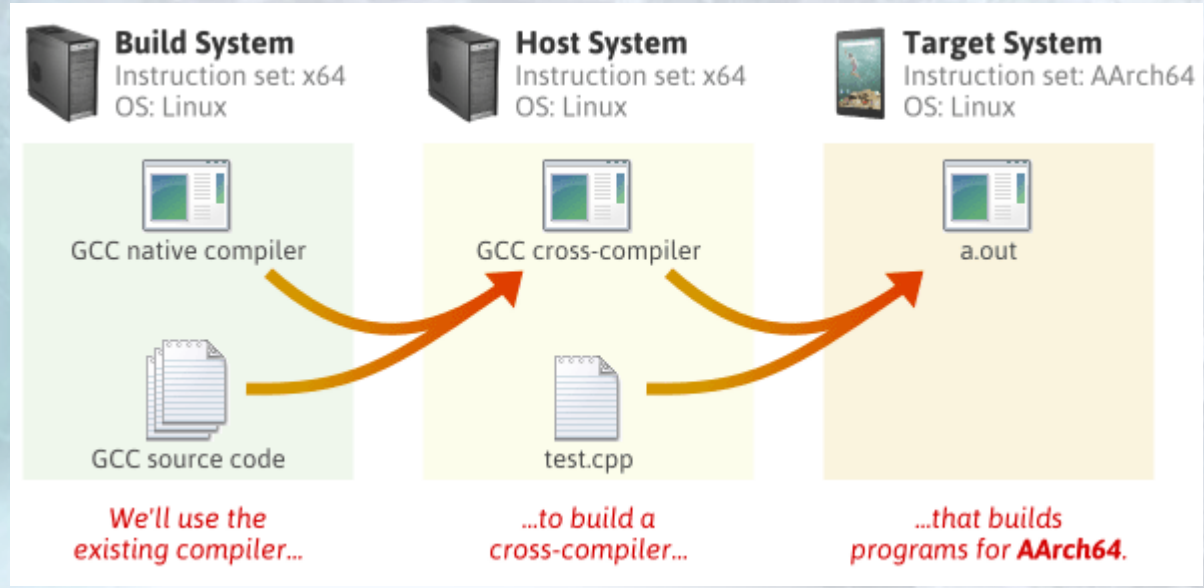
readelf	Reads the **contents** of an ELF File (**object file or executable**)
objdump	Reads the internals of an ELF file **including assembly code**
nm	Reads the **symbols** inside an object file or and executable
strings	Reads **text strings** inside a binary file
strip	**Strips** the **binary file** from **some optional sections**
addr2line	**Converts** an address in the binary to **a source file name and line number**
size	Display **the ELF file section sizes** and **total size**

gdb	Debugger

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Definition

- The **build machine**, where the toolchain is built.
- The **host machine**, where the toolchain will be executed.
- The **target machine**, where the binaries created by the toolchain are executed.



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Components – GCC

- To perform complete compilation process of the main.c source
  - **$ gcc main.c**
    - This command will generate the a.out executable (assuming the program is composed of a single file)
- To change the name of the generated executable
  - **$ gcc main.c -o test**
    - This command will generate an executable with the name test
- To specify the Include path (path for header files)
  - **$ gcc main.c -I/usr/share/include -I. -I./inc/ -I../../inc**
- To enable all warning during compilation process
  - **$ gcc -Wall main.c -o test**
- To convert warnings into errors
  - **$ gcc -Wall -Werror main.c -o test**
- To pass options to gcc in a file (instead of in the command)
  - **$ gcc main.c @options-file**
    - Where options-file is a text file that will contain the required options

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

GCC Partial Compilation

This will only generate main.i file which is a pre-processed source file

main.i (Embedded_Linux /media/embedded_system_ks/

```
856 # 2 "main.c" 2
857
858
859 # 3 "main.c"
860 int gval1 ;
861 int main()
862 {
863     int lval2 ;
864     printf ("hello (learn-in-deptj.com \n");
865     return 0 ;
```

main.s (Embedded_Linux /media/embedded_system_ks/Embedded_Linux/LAB1)

```
 8          .type    main, @function
 9 main:
10 .LFB0:
11          .cfi_startproc
12          leal     4(%esp), %ecx
13          .cfi_def_cfa 1, 0
14          andl     $-16, %esp
15          pushl    -4(%ecx)
16          pushl    %ebp
17          .cfi_escape 0x10,0x5,0x2,0x75,0
18          movl     %esp, %ebp
```

main.c (Embedded_Linux /media/embedded_system_ks/Em

```
1 #include "stdio.h"
2
3 int gval1 ;
4 int main()
5 {
6     int lval2 ;
7     printf ("hello (learn-in-deptj.com \n");
8     return 0 ;
9 }
```

embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_Linux/LAB1

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ gedit main.c &
[1] 3885
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ gcc -E main.c > main.i
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ gcc -S main.c > main.s
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ gcc -c main.c
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ gcc -save-temps main.c -o test
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$ gedit main.i main.s &
[1] 3950
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB1$
```

This will save:

- File after pre-processing (main.i)
- File after compilation (main.s)
- File after assembly (main.o)
- Executable (test)

This will generate an assembly file main.s

This will generate a relocatable object file main.o

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Creating a Static Library (ar Command)

- **$ ar [options] <library name> <object files>**
  - library object files should not contain **a main function**
  - Static library names should should start with "lib" and end with ".a" such as: libmath.a , libcontrol.a, libvision.a
- To create a static library file
  - $ ar rcs libmylib.a file1.o file2.o
- To add another object file to the library
  - $ ar r libmylib.a file3.o
- To remove an object file from the library
  - $ ar d libmylib.a file3.o
- To view the object files in the library
  - $ ar t libmylib.a
- To extract object files from the library
  - $ ar x libmylib.a

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Creating a Static Library (ar Command) example

myPrint_API.c

```
#include "myPrint_API.h"

  void learn_in_depth(char* buff)
{
        printf ("%s",buff);
}
```

myPrint_API.h

```
extern void learn_in_depth(char* buff);
#include "stdio.h"
```

main.c

```
#include "myPrint_API.h"

int gval1 ;
int main()
{
   int lval2 ;
   learn_in_depth ("hello (learn-in-deptj.com \n");
   return 0 ;
```

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ gcc -c myPrint_API.c
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ gcc -c main.c
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ ls *.o
main.o  myPrint_API.o
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ ar rcs lib_learn_in_depth.a myPrint_API.o
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ gcc main.c -I . lib_learn_in_depth.a
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ ./a.out
hello (learn-in-deptj.com
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ ar t lib_learn_in_depth.a
myPrint_API.o
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ ar x lib_learn_in_depth.a
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/LAB0$ ls
lib_learn_in_depth.a  main.c  main.o  myPrint_API.c  myPrint_API.h  myPrint_API.o
```

To view the object files in the library

To extract object files from the library

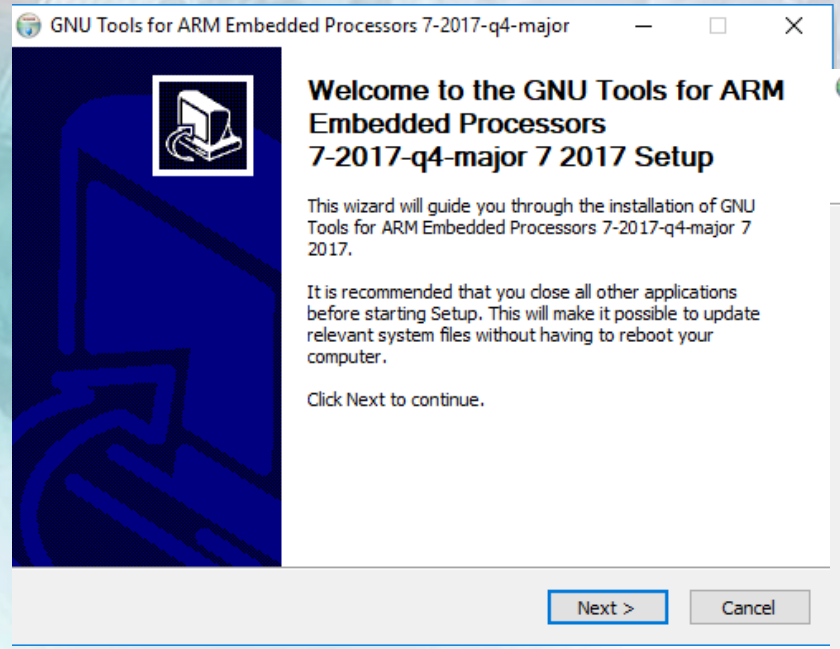
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

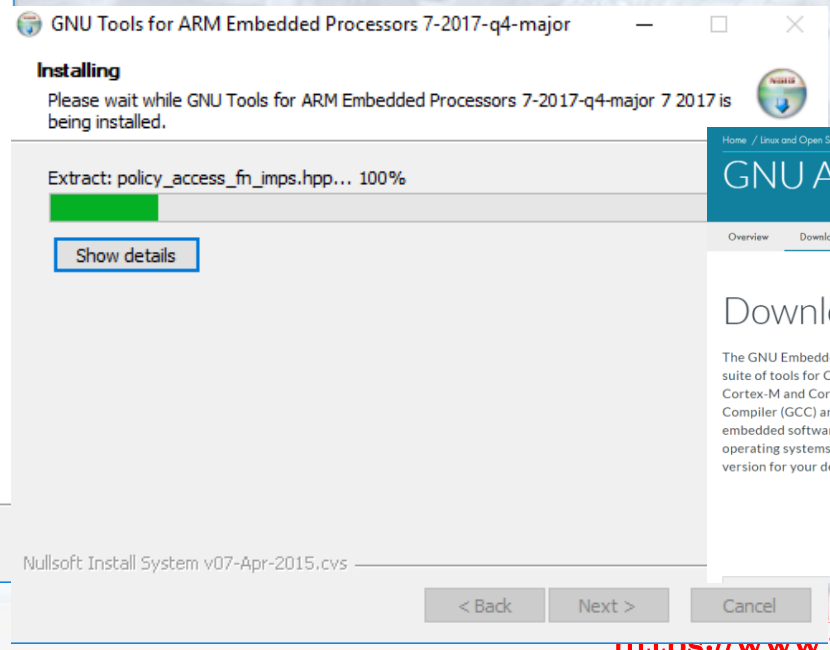
Most used ARM Cross-toolchain

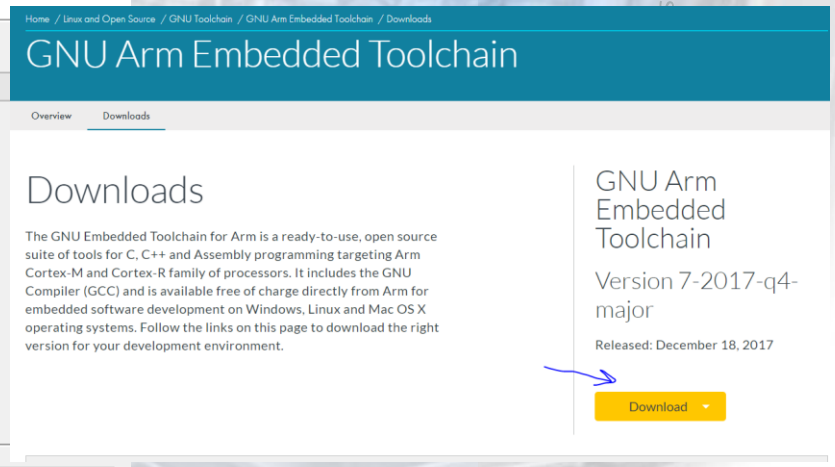
- GNU GCC for ARM embedded Processors (free and open source)
- armcc from ARM ltd. (ships with KEIL, requires licensing)

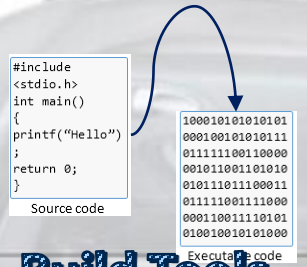
We will use prebuilt toolchain

- Download GNU ARM toolchain
  - https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads









Build Tools

- Compiler,Linker,assembler
  - arm-non-eabi-gcc
- linker
  - Arm-non-eabi-ld

#LEARN_IN DEPTH

#Be_professional_in embedded_system

Compilation Process

BE READY, THIS IS IMPORTANT PART

MOST OF THE INTERVIEW QUESTIONS IS CONCENTRATED HERE

LEARN-IN-DEPTH

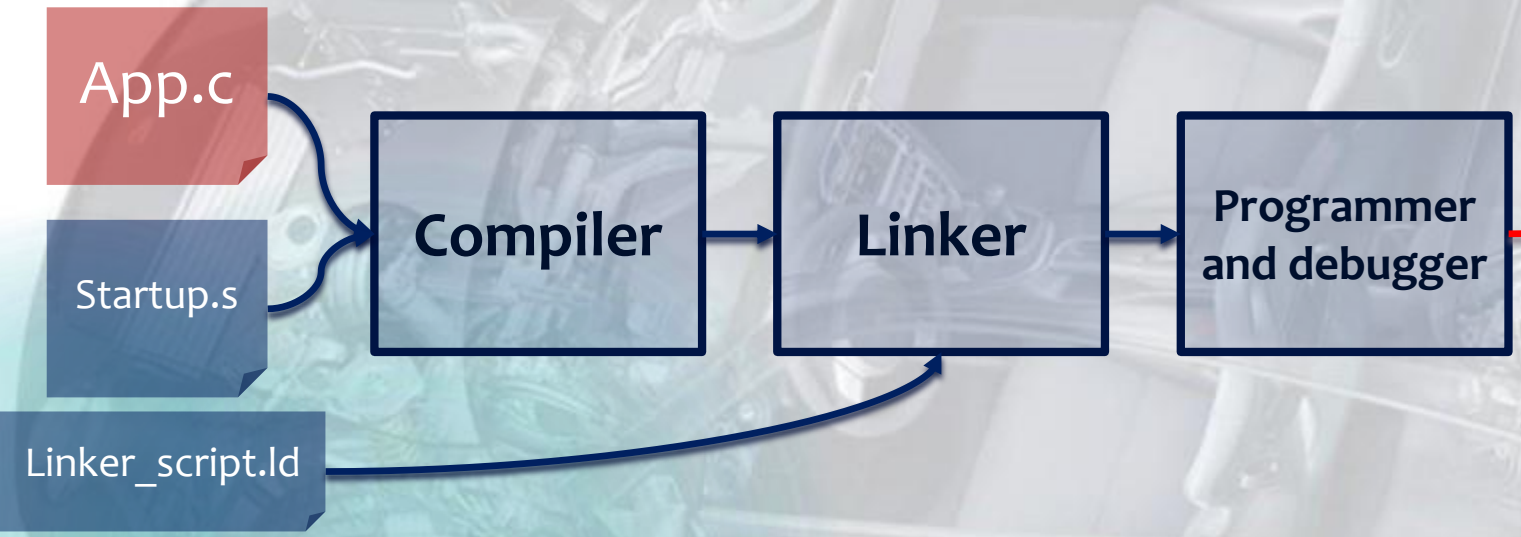
Be professional in embedded system

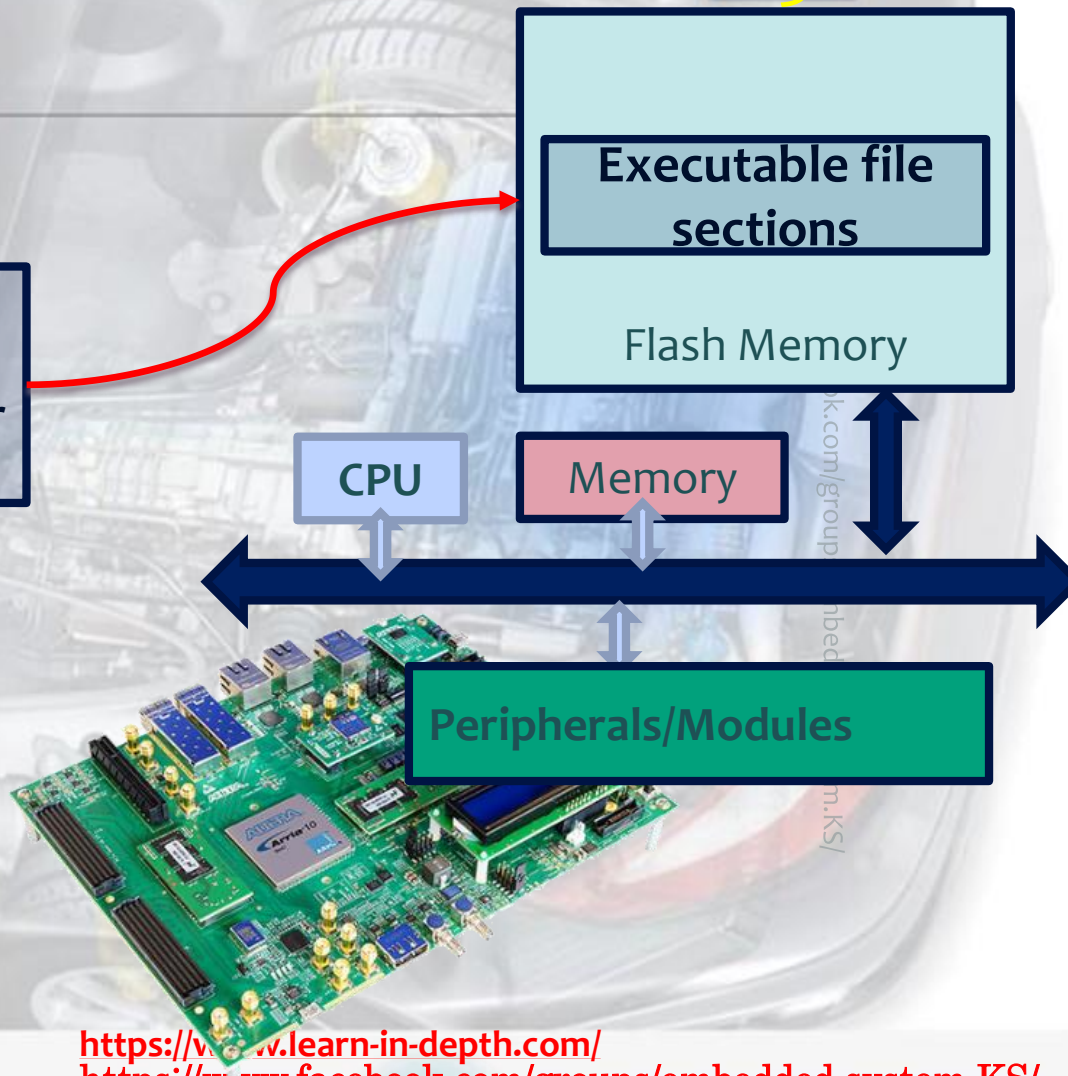
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

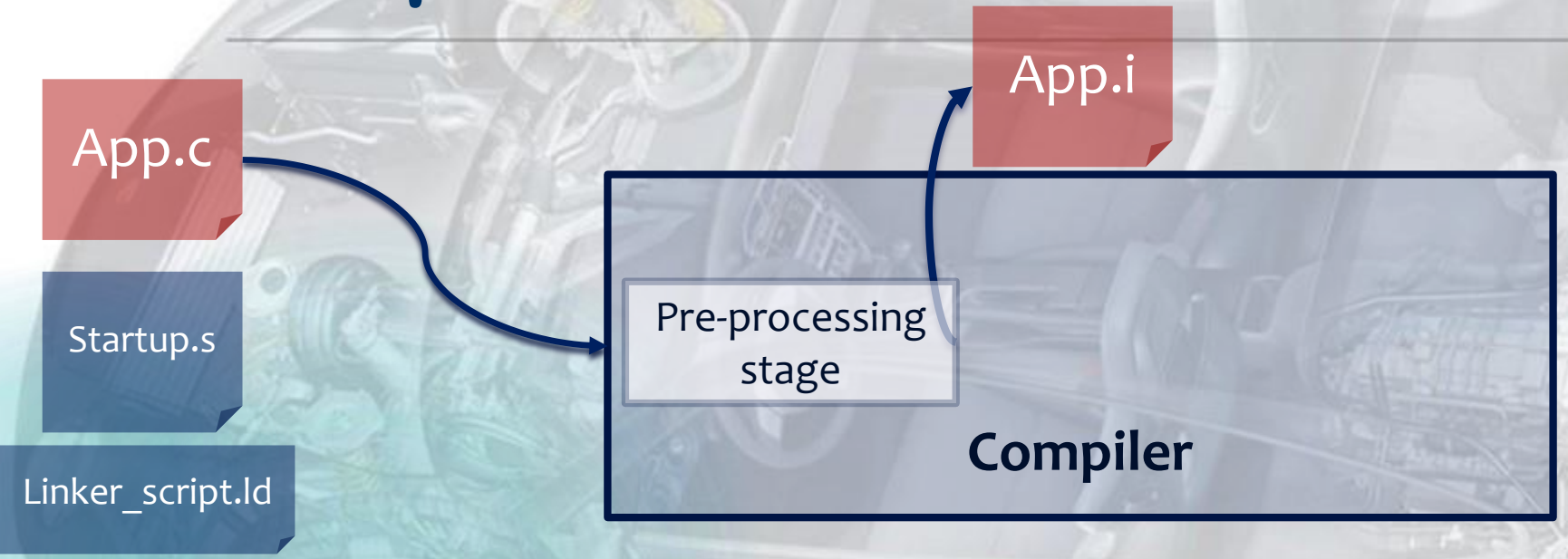
Compilation Process





https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Compilation Process Cont.

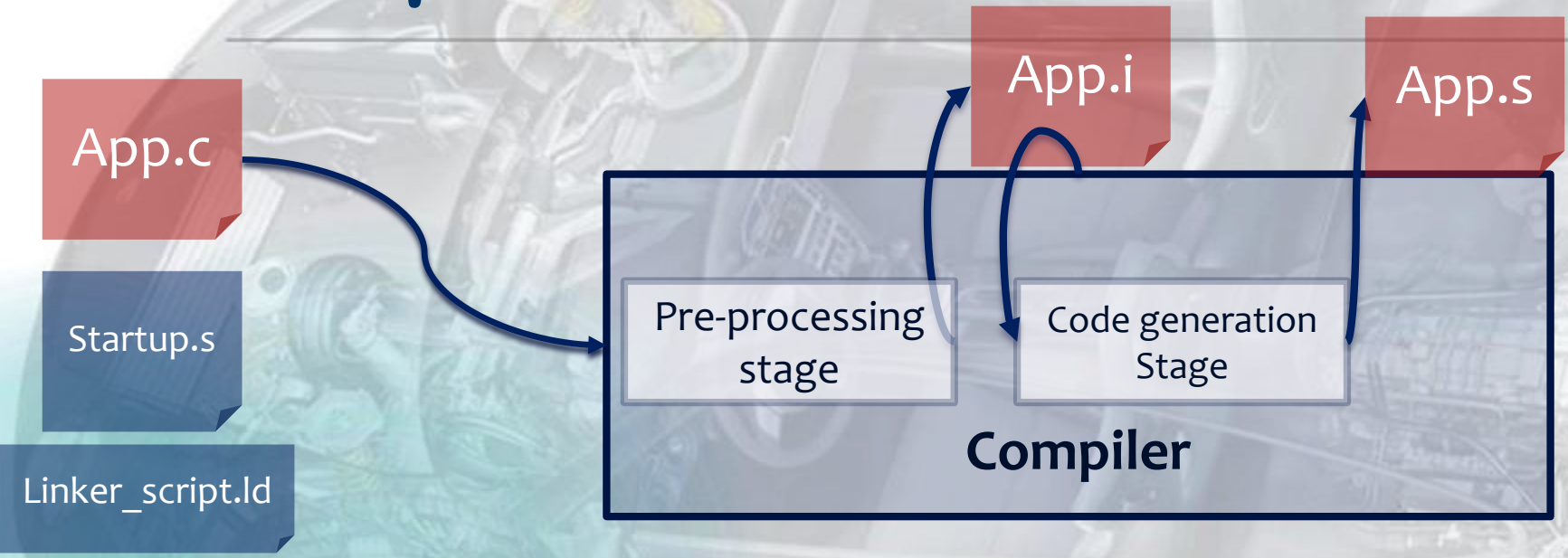


All pre-processing directives will be resolved

Macros
#include
#ifdef,… etc

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Compilation Process Cont.

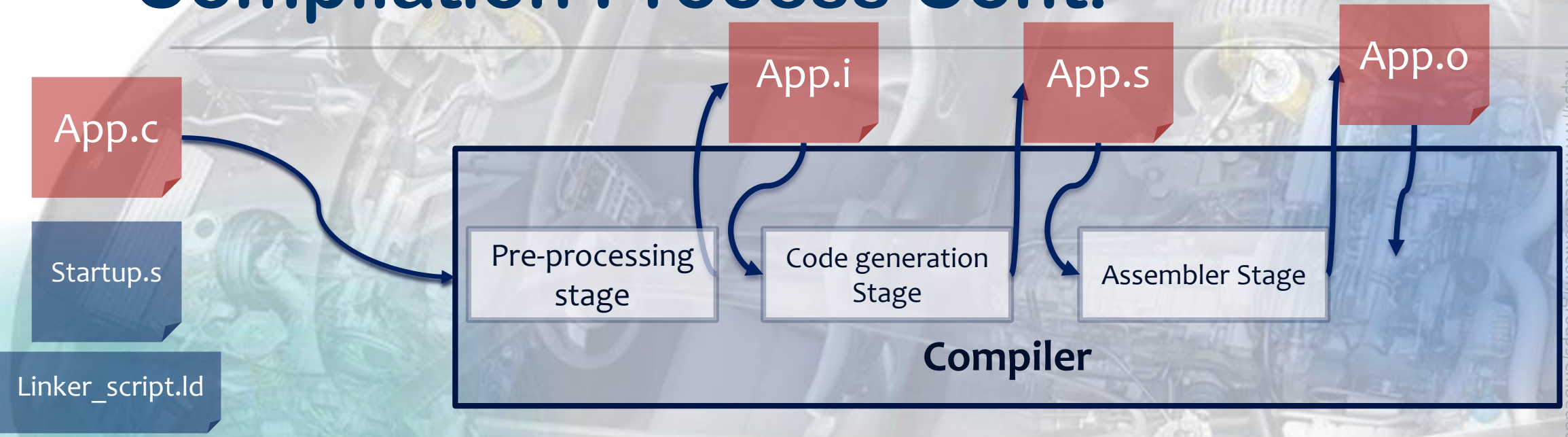


All pre-processing directives will be resolved
Macros
#include
#ifdef,… etc

Higher Level language code c will
Be converted into processor architecture level assembly

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Compilation Process Cont.



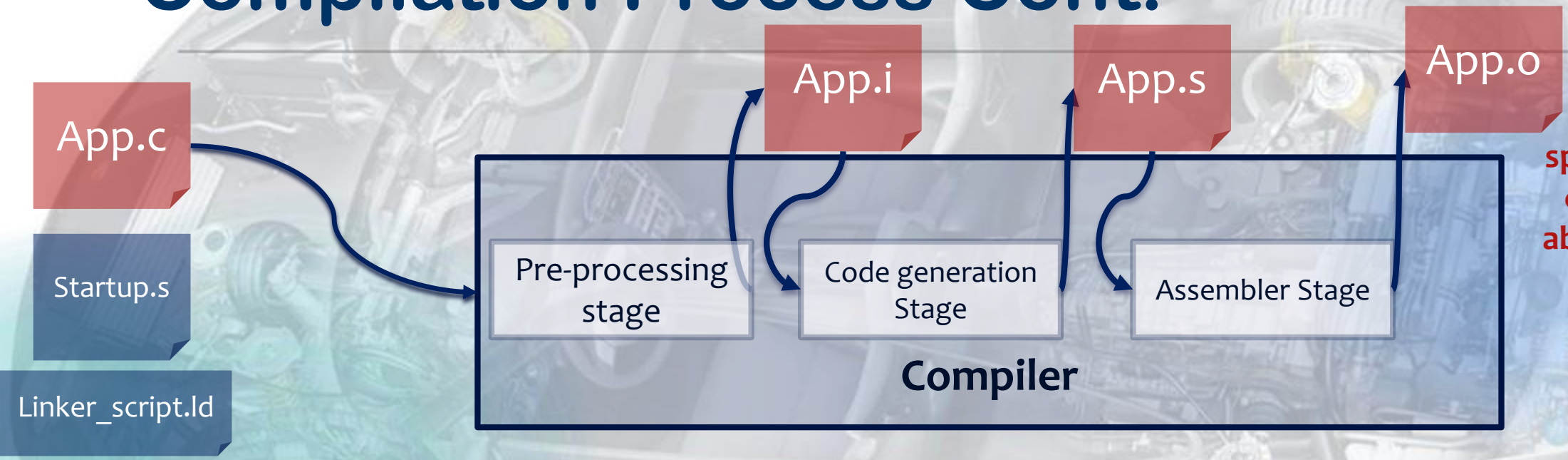
All pre-processing directives will be resolved
Macros
#include
#ifdef,... etc

Higher Level language code c will
Be converted into processor architecture level assembly

Assembly will be converted into opcodes & operands (binary)

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Compilation Process Cont.



Processor architecture specific machine codes with no absolute address (relocatable)

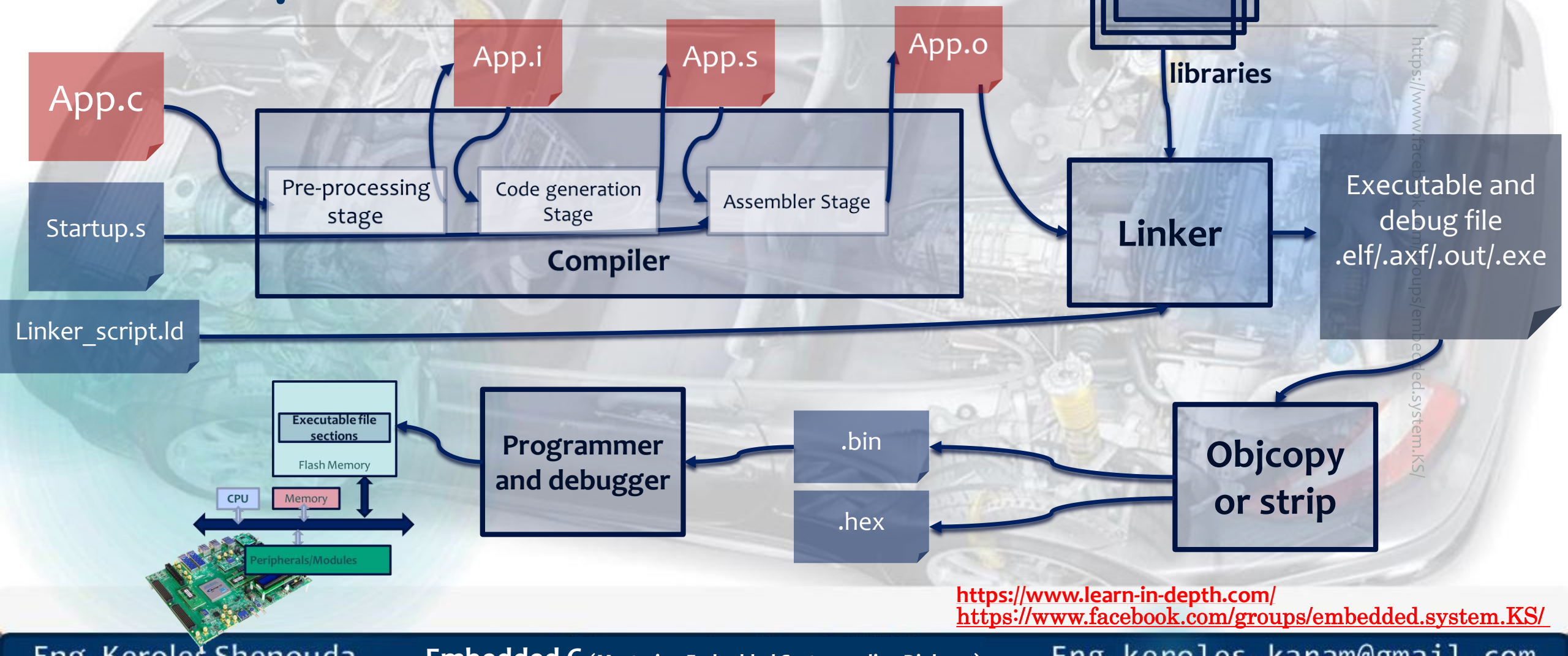
All pre-processing directives will be resolved
Macros
#include
#ifdef,… etc

Higher Level language code c will Be converted into processor architecture level assembly

Assembly will be converted into opcodes & operands (binary)

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

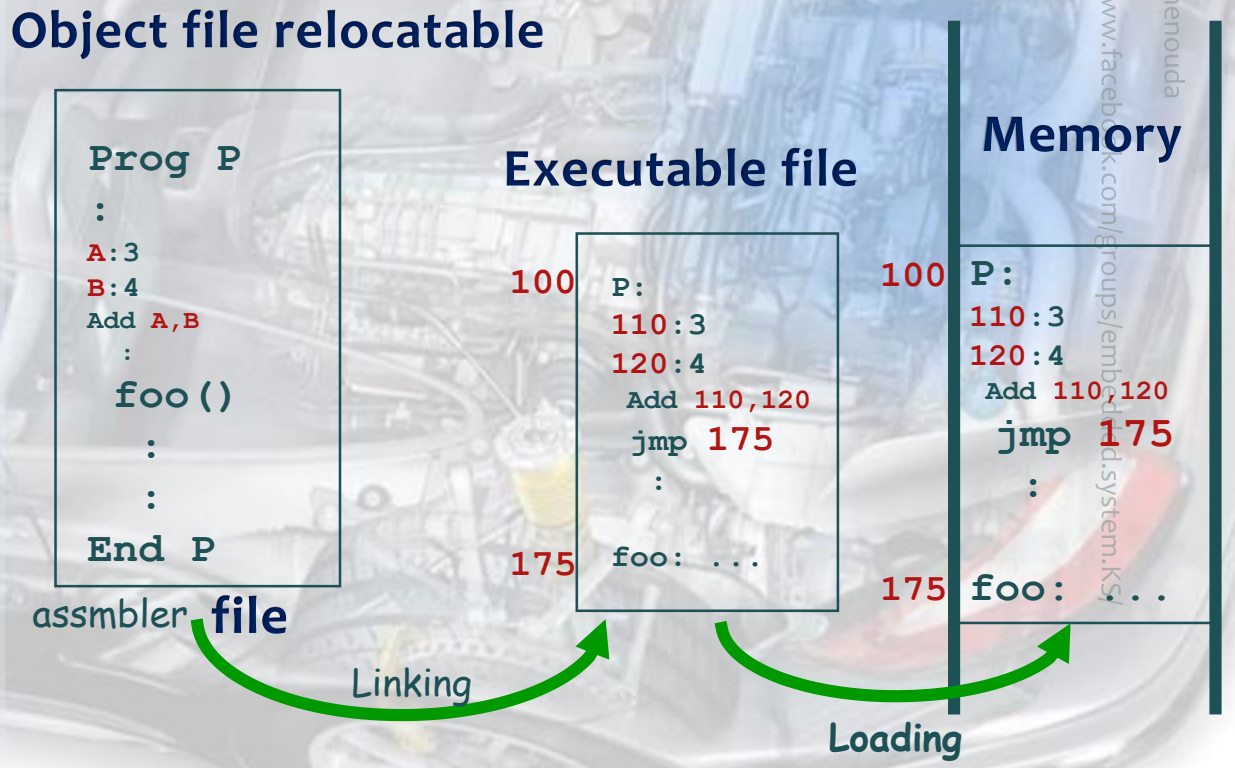
Compilation Process Cont.



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Compile Time Binding

- if memory **location** is **fixed** and **known** at compile time, absolute code with absolute address can be generated.
- Must Recompile Code if starting location changed.



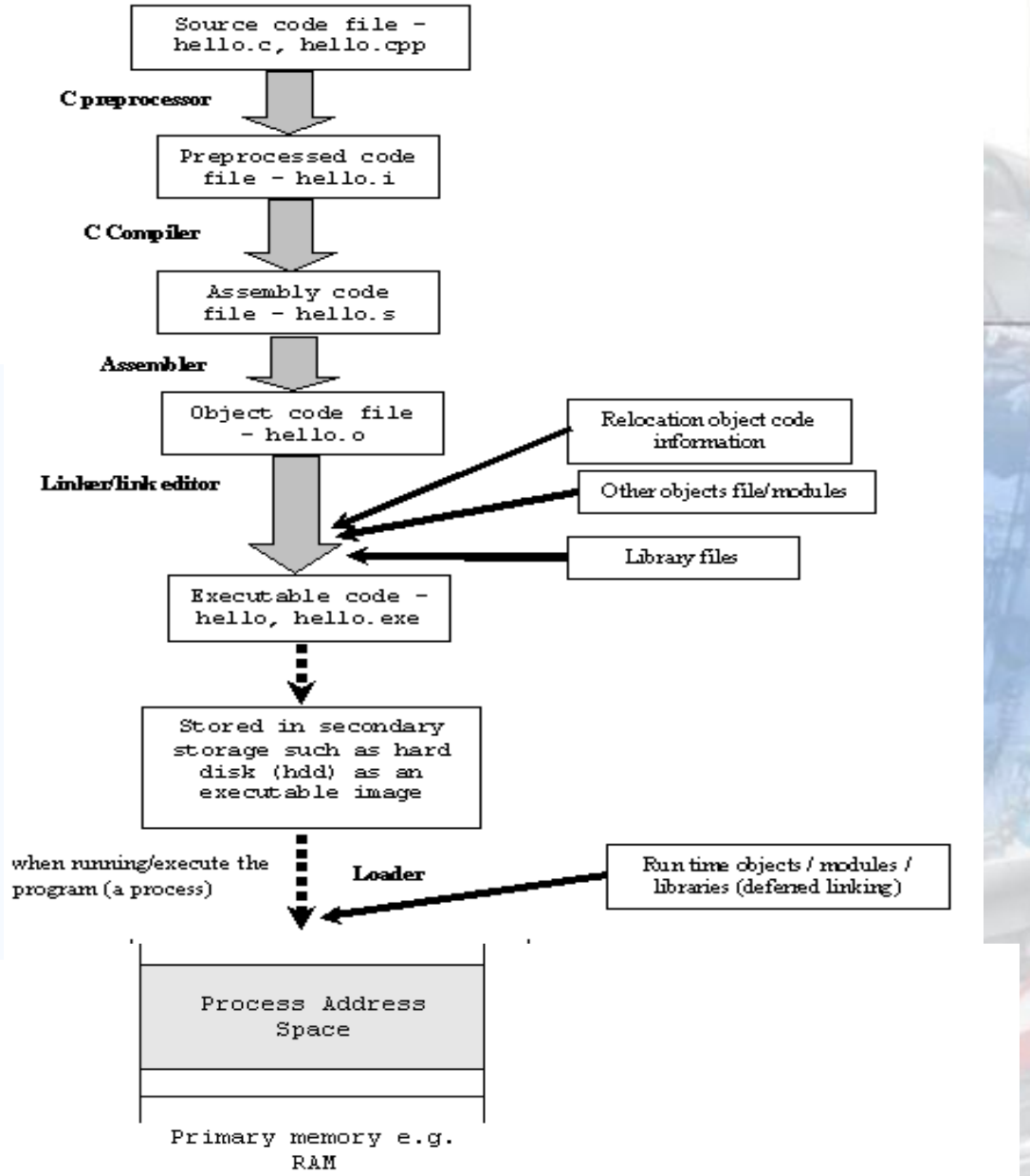
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Summary

- Preprocessing (Preprocessor)
  - It processes include files, conditional compilation instructions and macros.
  - Command: $cpp hello.c hello.i
  - hello.c is source program, hello.i is ASCII intermediate code
- Compiling (Compiler)
  - It takes the output of the preprocessor and generates assembler source code
  - Command: $cc hello.i -o hello.s
- Assembly (Assembler)
  - It takes the assembly source code and produces an assembly listing with offsets.
  - The assembler output is stored in an object file.
  - Command:$as -o hello.o hello.s
- Linking (Linker)
  - It takes one or more object files or libraries as input and combines them to produce a single (usually executable) file.
  - Linux command for linker is ld

Summary Cont.

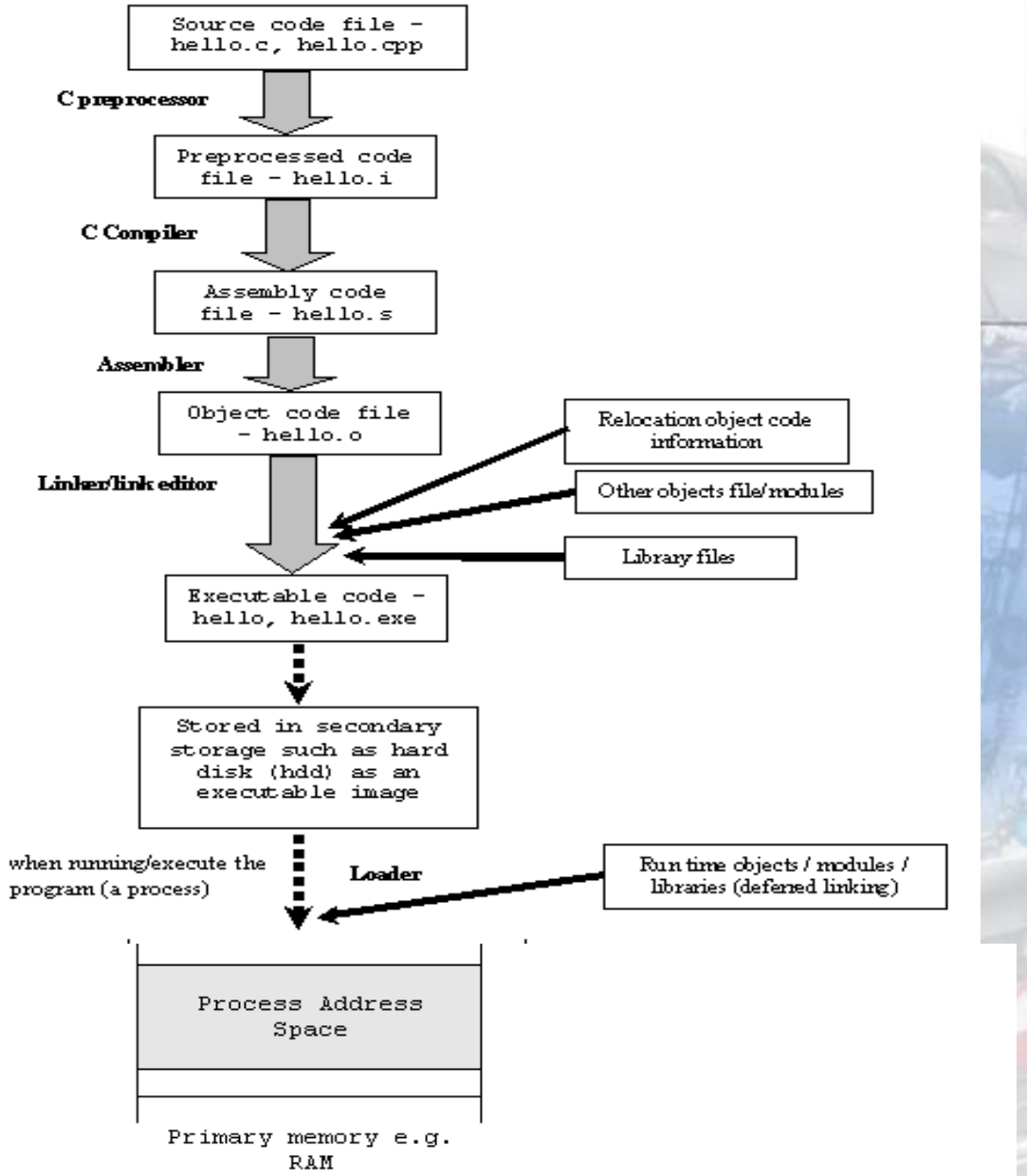
- **Preprocessing** It is the first stage of compilation. It processes preprocessor directives like include-files, conditional compilation instructions and macros.
- **Compilation** It is the second stage. It takes the output of the preprocessor with the source code, and generates assembly source code.



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

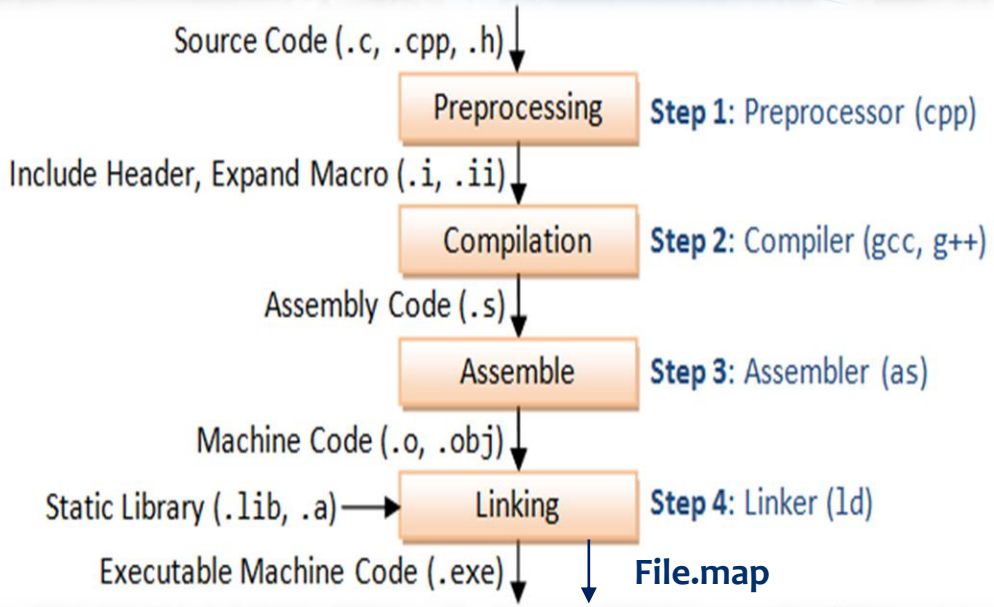
Summary Cont.

- **Assembler stage** It is the third stage of compilation. It takes the assembly source code and produces the corresponding object code.
- **Linking It is the final stage** of compilation. It takes one or more object files or libraries and **linker script as input** and combines them to **produce a single executable file**. In doing so, **it resolves references to external symbols, assigns final addresses to procedures/functions and variables**, and revises code and data to reflect new addresses (a process called relocation).



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

- The **.map** file gives a complete listing of all code and data addresses for the final software image.
- It provides information similar to the contents of the linker script described earlier. However, these are results rather than instructions and therefore include the actual lengths of the sections and the names and locations of the public symbols found in the relocatable program.



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

- **linker** can be passed to it in the form of a *linker script*. Such scripts are sometimes used to control the exact order of the code and data sections within the relocatable program

```
ENTRY (main)

MEMORY
{
    ram : ORIGIN = 0x00400000, LENGTH = 64M
    rom : ORIGIN = 0x60000000, LENGTH = 16M
}

SECTIONS
{
    data :                                  /* Initialized data. */
    {
        _DataStart = . ;
        *(.data)
        _DataEnd   = . ;

    } >ram

    bss :                                   /* Uninitialized data. */
    {
        _BssStart = . ;
        *(.bss)
        _BssEnd   = . ;
    } >ram

    text :                                  /* The actual instructions. */
    {
        *(.text)
    } >ram
}
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Resolving Symbols DURING LINKING

Global

Global

External

Local

```
int buf[2] = {1, 2};

int main()
{
  swap();
  return 0;
}
```

main.c

External

Linker knows nothing of temp

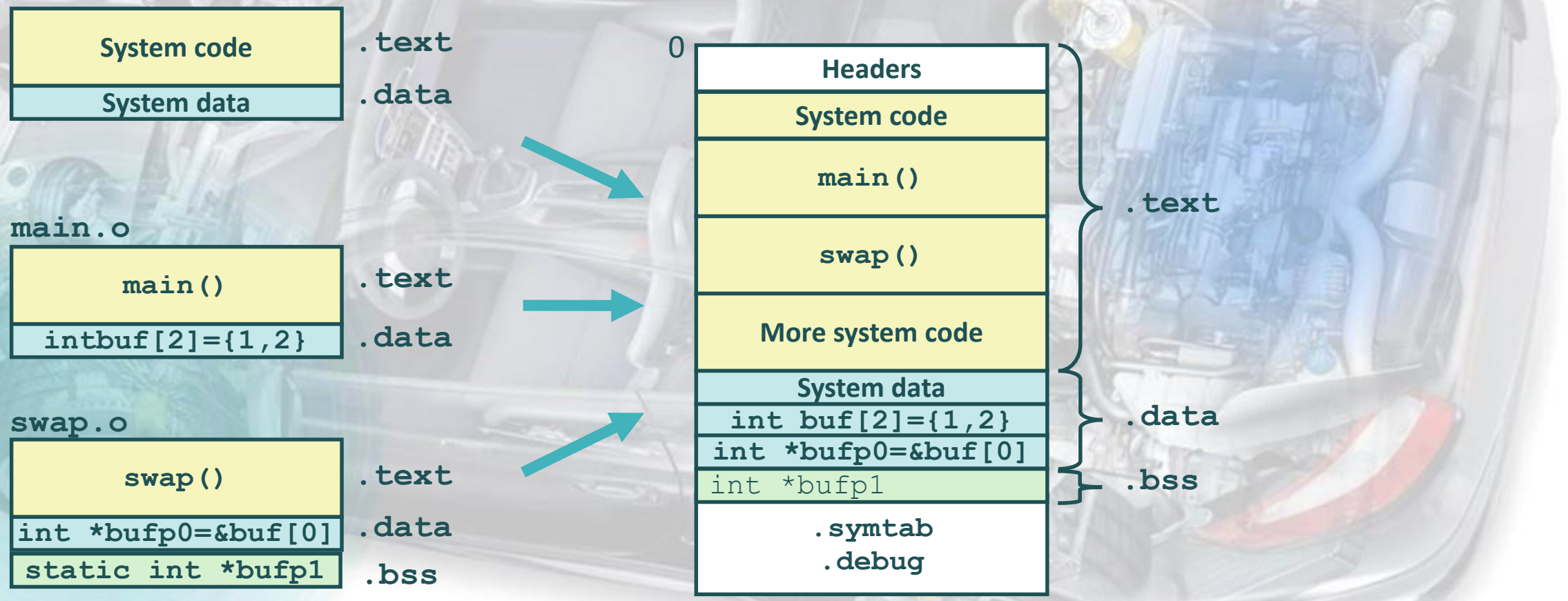
```
extern int buf[];

Int *bufp0 = &buf[0];
static int *bufp1;

void swap()
{
int temp;

  bufp1 = &buf[1];
  temp = *bufp0;
  *bufp0 = *bufp1;
  *bufp1 = temp;
}
```

Global

swap.c

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Relocating Code and Data DURING Linking



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

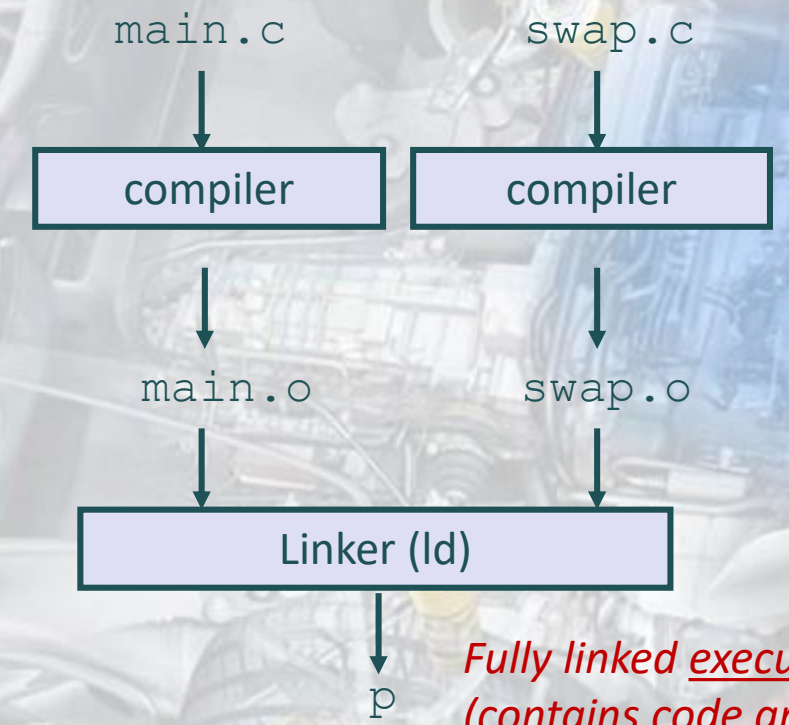
Example C Program

main.c

```
int buf[2] = {1, 2};

int main()
{
  swap();
  return 0;
}
```

swap.c

```
extern intbuf[];

int*bufp0 = &buf[0];
static int *bufp1;

void swap()
{
int temp;

  bufp1 = &buf[1];
  temp = *bufp0;
  *bufp0 = *bufp1;
  *bufp1 = temp;
}
```



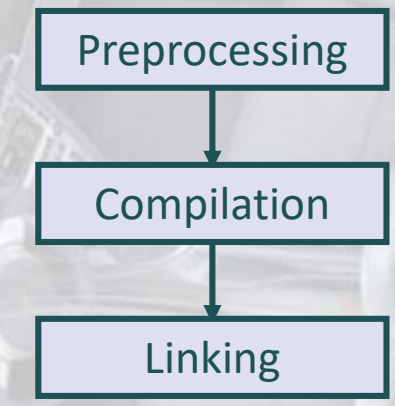
Source files

Separately compiled relocatable object files

Fully linked executable object file (contains code and data for all functions defined in `main.c and swap.c`)

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Arm-non-eabi-gcc



Compilation flags

https://gcc.gnu.org/onlinedocs/gcc-9.3.0/gcc/

```
kkhalil@egc-kkhalil-lt MINGW64 /d/courses/new_diploma/C/diploma4/ARM/bin
$ arm-none-eabi-gcc.exe --help
Usage: arm-none-eabi-gcc.exe [options] file...
Options:
  -pass-exit-codes         Exit with highest error code from a phase.
  --help                   Display this information.
  --target-help            Display target specific command line options.
  --help={common|optimizers|params|target|warnings|[^]{joined|separate|undocumented}}[,...].
                           Display specific types of command line options.
  (Use '-v --help' to display command line options of sub-processes).
  --version                Display compiler version information.
  -dumpspecs               Display all of the built in spec strings.
  -dumpversion             Display the version of the compiler.
  -dumpmachine             Display the compiler's target processor.
  -print-search-dirs       Display the directories in the compiler's search path.
  -print-libgcc-file-name  Display the name of the compiler's companion library.
  -print-file-name=<lib>   Display the full path to library <lib>.
  -print-prog-name=<prog>  Display the full path to compiler component <prog>.
  -print-multiarch         Display the target's normalized GNU triplet, used as
                           a component in the library path.
  -print-multi-directory   Display the root directory for versions of libgcc.
  -print-multi-lib         Display the mapping between command line options and
                           multiple library search directories.
  -print-multi-os-directory Display the relative path to OS libraries.
  -print-sysroot           Display the target libraries directory.
  -print-sysroot-headers-suffix Display the sysroot suffix used to find headers.
  -Wa,<options>            Pass comma-separated <options> on to the assembler.
  -Wp,<options>            Pass comma-separated <options> on to the preprocessor.
  -Wl,<options>            Pass comma-separated <options> on to the linker.
  -Xassembler <arg>        Pass <arg> on to the assembler.
  -Xpreprocessor <arg>     Pass <arg> on to the preprocessor.
  -Xlinker <arg>           Pass <arg> on to the linker.
  -save-temps              Do not delete intermediate files.
  -save-temps=<arg>        Do not delete intermediate files.
  -no-canonical-prefixes   Do not canonicalize paths when building relative
                           prefixes to other gcc components.
  -pipe                    Use pipes rather than intermediate files.
  -time                    Time the execution of each subprocess.
  -specs=<file>            Override built-in specs with the contents of <file>.
  -std=<standard>          Assume that the input sources are for <standard>.
  --sysroot=<directory>    Use <directory> as the root directory for headers
                           and libraries.
  -B <directory>           Add <directory> to the compiler's search paths.
  -v                       Display the programs invoked by the compiler.
  -###                     Like -v but options quoted and commands not executed.
  -E                       Preprocess only; do not compile, assemble or link.
  -S                       Compile only; do not assemble or link.
  -c                       Compile and assemble, but do not link.
  -o <file>                Place the output into <file>.
  -pie                     Create a position independent executable.
  -shared                  Create a shared library.
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

#LEARN_IN DEPTH

#Be_professional_in embedded_system

Live Demo

LEARN-IN-DEPTH

Be professional in

embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

#LEARN_IN DEPTH

#Be_professional_in embedded_system

Next Lecture is very important

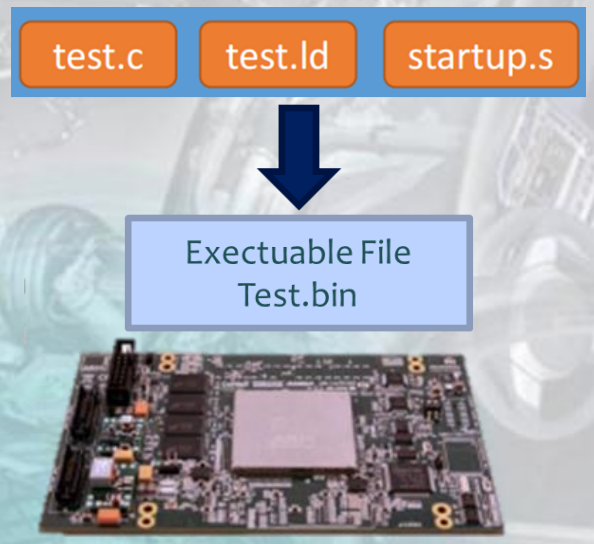
BE READY ;)

LEARN-IN-DEPTH

Be professional in embedded system

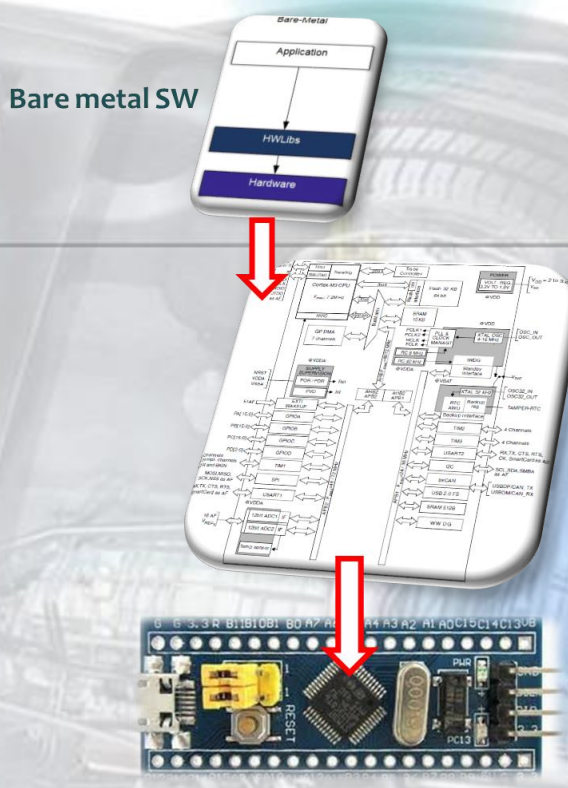
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

In next Lecture you will learn



Lab1 on ARM VersatilePB

You will Learn. Startup, linker, location counter, linker script symbols, Makefile, GDB commands Binary utilities: Objdump, strip, addr2line, size, readelf



Lab2 on ARM CortexM3 STM32

You will Learn. Complex Startup, Complex linker, Complex Makefile, linking flags, Analyzing executable file, analyzing relocatable obj files , executable sections, linker & locator and Debugger circuits.

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Thank you!

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

References

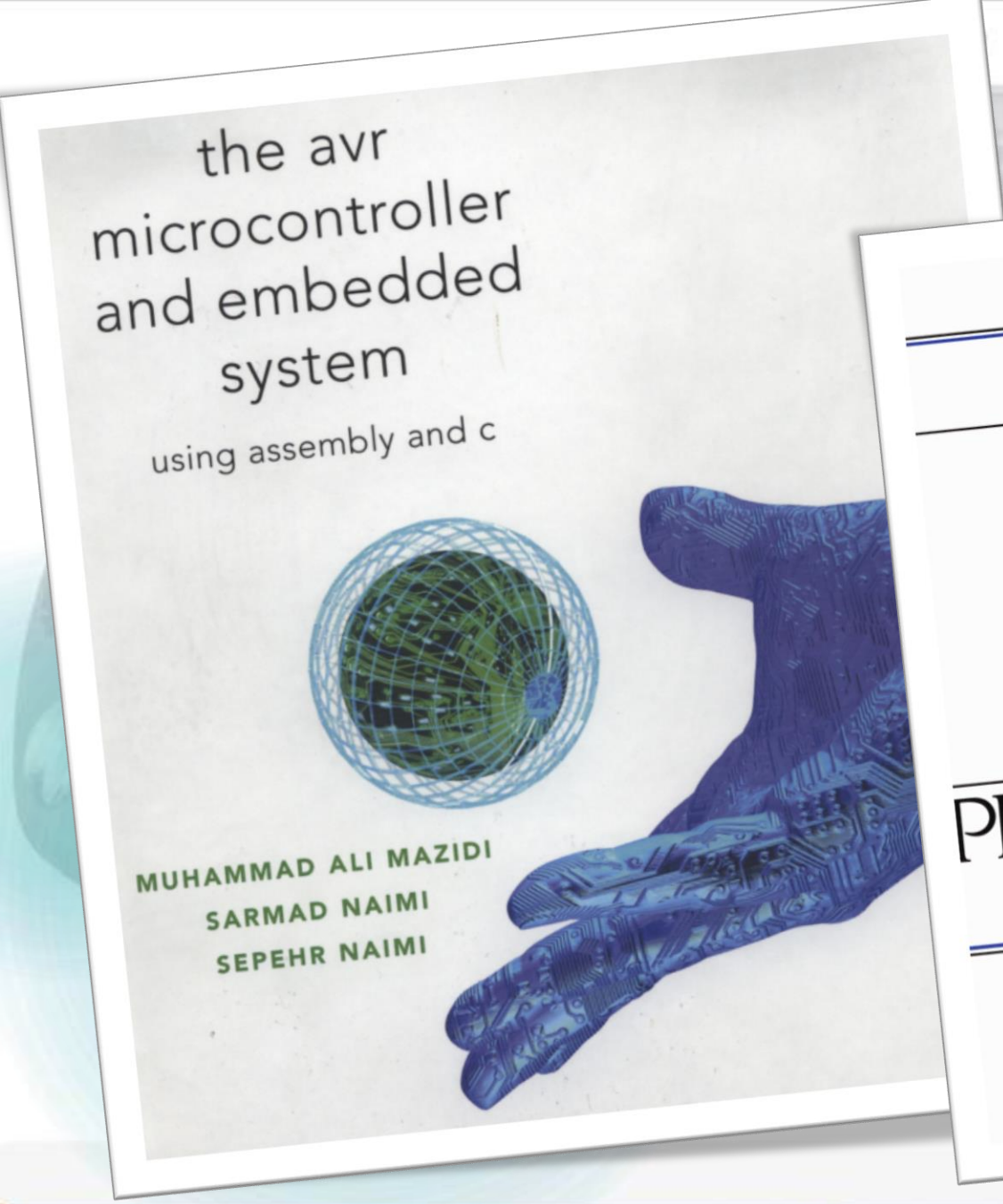
- **Freedom Embedded**
  Balau's technical blog on open hardware, free software and security
  - https://balau82.wordpress.com
- AUTOSAR Specification for data types

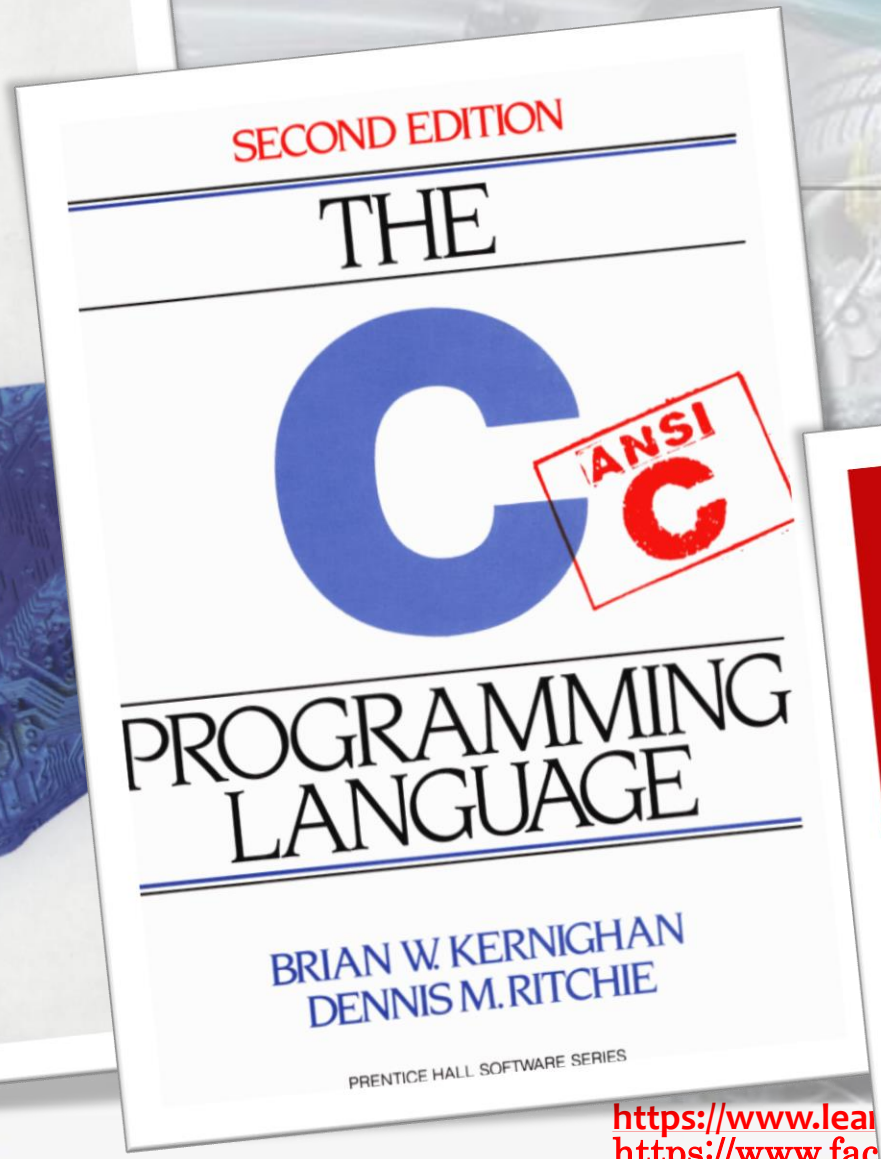
#LEARN_IN DEPTH

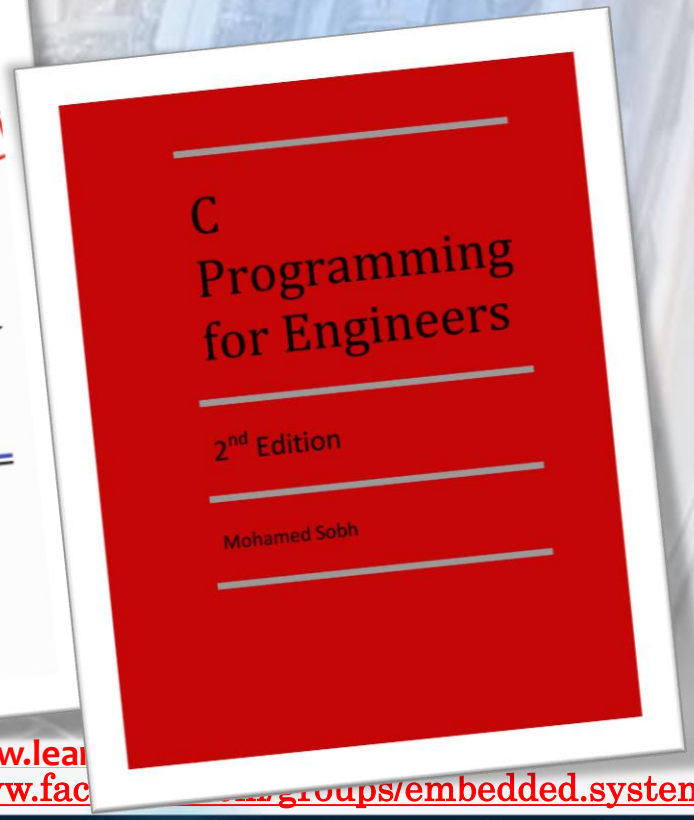
#Be_professional_in embedded_system

useful references for

C AND EMBEDDED C







https://www.lea
https://www.fac groups/embedded.system.KS/

References Cont.

- Udemy Courses:
  - 1) Microcontroller Embedded C Programming: absolute beginners**(Embedded C)**
  - 2) Embedded Systems Programming on ARM Cortex-M3/M4 Processor**(ARM Cortex M4 Processor specific)**
  - 3) Mastering Microcontroller with Embedded Driver Development**(MCU1)**
  - 4) Mastering Microcontroller: TIMERS, PWM, CAN, RTC,LOW POWER**(MCU2)**
  - 5) Mastering RTOS: Hands-on FreeRTOS and STM32Fx with Debugging**(RTOS)**
  - 6) ARM Cortex M Microcontroller DMA Programming Demystified**(DMA)**
- Memory Management article
- Introduction to Operating Systems Class 9 - Memory Management
- Virtual Memory lecture for Introduction to Computer Architecture at Uppsala University.
- OS Lecture Note 13 - Address translation, Caches and TLBs
- CSE 331 Operating Systems Design lectures
- CSE 331 Virtual Memory
- CSE 332 Computer Organization and Architecture
- File Systems: Fundamentals.
- Linux System Programming
- System Calls, POSIX I/O
  CSE 333 Spring 2019, Justin Hsia
- Linux basic commands

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

References

- Embedded Linux training
  - https://bootlin.com/training/
- Linux kernel and driver development training
- Yocto Project and OpenEmbedded development training
- Buildroot development training
- Building Embedded Linux Systems, 2nd Edition
- Mastering Embedded Linux Programming - Second Edition
- Christopher Hallinan, **Embedded Linux Primer,** Prentice Hall, 2006.
- **Silberschatz**, Galvin, and Gagne's **Operating System Concepts**, **Eighth Edition**.
- Robert love, linux kernel develoepemnt 3 rd edition 2010
- **Advanced Linux Programming By Mark Mitchell, Jeffrey Oldham, Alex Samuel**
- **Linux OS in Embedded Systems & Linux Kernel Internals(2/2)**
  - Memory Management, Paging, Virtual Memory, File system and its implementation, Secondary Storage(HDD), I/O systems
- Ahmed ElArabawy Courses at http://linux4embeddedsystems.com/

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

References Cont.

- File Permissions in Linux/Unix with Example
- Introduction to UNIX / Linux - 4
- 12.2 Basic I/O Concepts
- Communicating with Hardware
- I/O Systems I/O Hardware Application I/O Interface
- COMPUTER ARCHITECTURE
- OS
- Linux Operating System
- W4118 Operating Systems, Instructor: Junfeng Yang
- CSNB334 Advanced Operating Systems 4. Process & Resource Management.
- Using the POSIX API
- Linux Memory Management

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

References Cont.

- Chapter 2: Operating-System Structures
- **POSIX Threads Programming**
- Pthreads: A shared memory programming model
  https://slideplayer.com/slide/8734550/
- Signal Handling in Linux Tushar B. Kute
- Introduction to Sockets Programming in C using TCP/IP
- **How to Run Two or More Terminal Commands at Once in Linux**
- **How to Build a GCC Cross-Compiler**