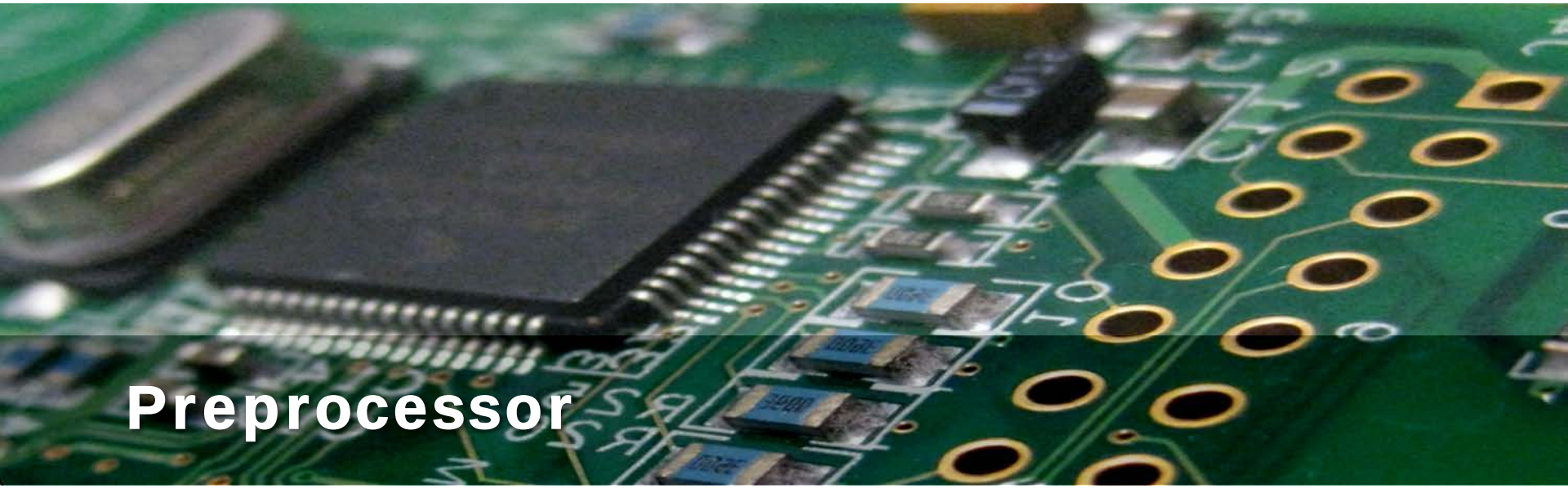
Lucerne University of
Applied Sciences and Arts
HOCHSCHULE
LUZERN
Technik & Architektur



Preprocessor

"There is always some prep work upfront."

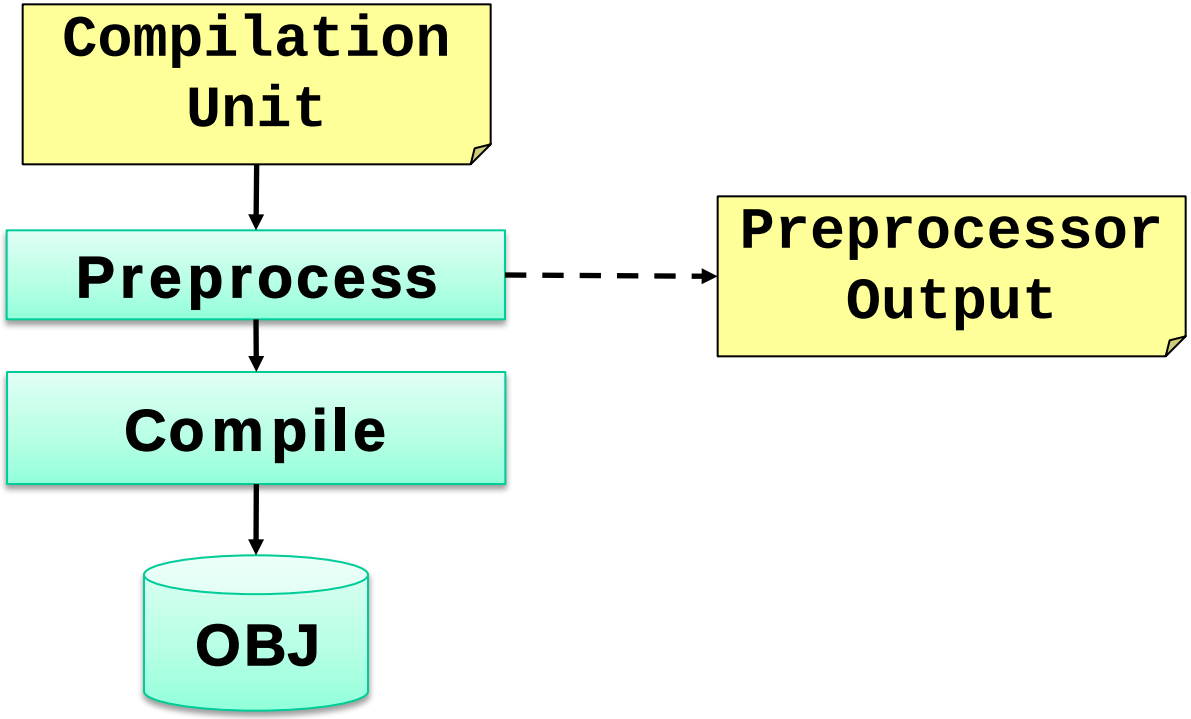
Prof. Erich Styger
erich.styger@hslu.ch
+41 41 349 33 01

Scriptum:
ANSI-C, Exploring Embedded C

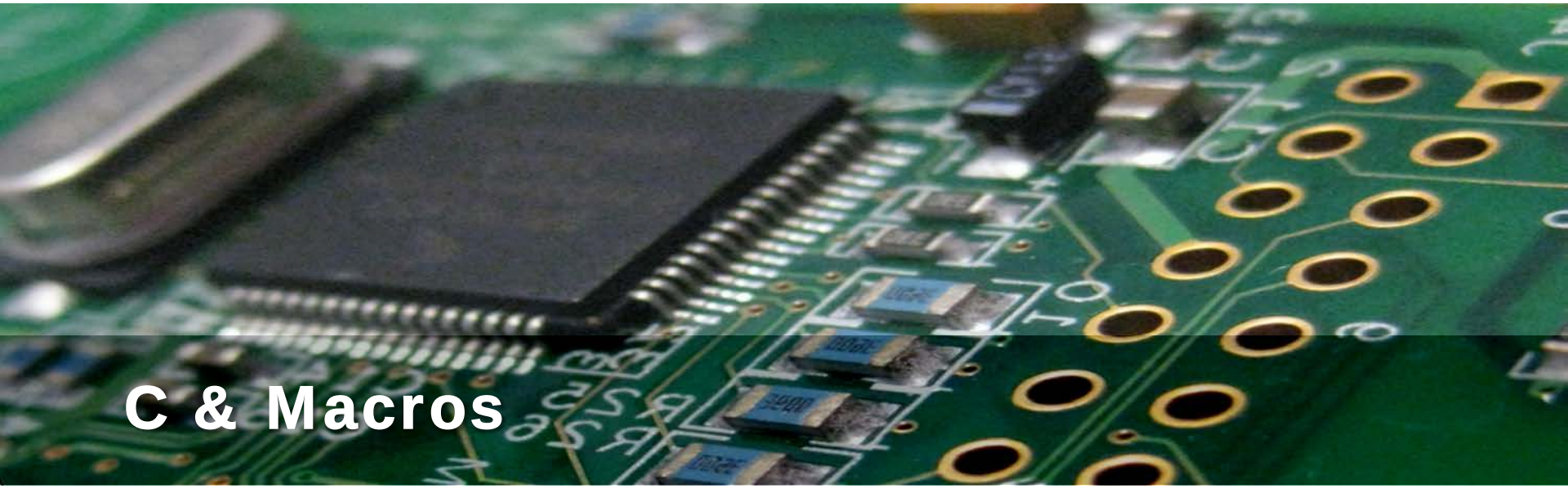
C/C++ Macros/#define

- Definition of a macro
- Compiler is replacing macros textually

```
#define BLUE   0
#define RED    1
#define YELLOW 2
```



Lucerne University of
Applied Sciences and Arts
HOCHSCHULE
LUZERN
Technik & Architektur



C & Macros

"Let's implement that application in C. That would keep things portable, hopefully. Just need to make sure that we get efficient code and reasonable speed out of it. Macros are a good idea. Might be a good idea to start with something simple: the LEDs!"

Prof. Erich Styger
erich.styger@hslu.ch
+41 41 349 33 01

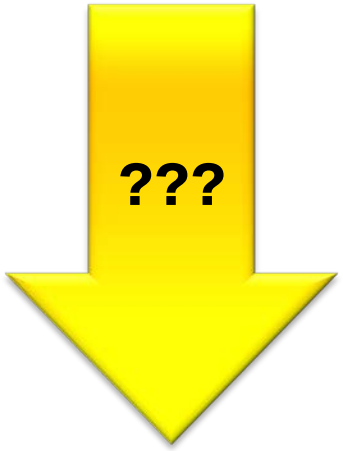
Learning Goals

- Problem: Using Macros

- Compiler

- Aspects of
  - Reusability
  - Optimization
  - Debugging
  - Maintenance

- Macros
  - Usage
  - Pitfalls



Textual Replacement

```
int ChangeColor(int color) {
  if (color == BLUE) {
    return RED;
  }
}
```

```
#define BLUE   0
#define RED    1
#define YELLOW 2
```

```
int ChangeColor (int color) {
  if (color == 0) {
    return 1;
  }
}
```

Why Macros?

- Names instead of ‚magic' numbers
  `#define DELAY_TIME_MS 10`
- Configuration
  `#define DEBUG_ME 1`
- Portability
  `#define ENABLE_INTERRUPT  _asm CLI;`
- Optimization

Knowing what you are doing
&
Be cautious with whatever you do

Traps & Pitfalls

```
#define INCI(i) {int a=0; i++;}

void main(void) {
  int a = 0, b = 0;
  INCI(a);
  INCI(b);
  printf("a is now %d, b is now %d\n", a, b);
}
```

```
void main(void) {
  int a = 0, b = 0;
  {int a=0; a++;};
  {int a=0; b++;};
  printf("a is now %d, b is now %d\n", a, b);
}
```

a is now 0, b is now 1

Traps & Pitfalls

```
#define PRE_DELAY   5
#define POST_DELAY  2
#define DELAY       PRE_DELAY + POST_DELAY
```

```
return totalDelay(int nofIterations) {
  return nofIterations * DELAY;
}
```

n * 5 + 2

Traps & Pitfalls

```
#define PRE_DELAY   5
#define POST_DELAY  2
#define DELAY       (PRE_DELAY + POST_DELAY)
```

```
#define PRE_DELAY   (5*3)
#define POST_DELAY  (2+5)
#define DELAY       ((PRE_DELAY) + (POST_DELAY))
```

```
return totalDelay(int nofIterations) {
  return nofIterations * (((5*3)) + ((2+5)));
}
```

LED's (Function Call)

```
typedef enum {
  LED_0 = (1<<0), /*!< Bit0 of port for LED0 */
  LED_1 = (1<<1), /*!< Bit1 of port for LED1 */
  LED_2 = (1<<2), /*!< Bit2 of port for LED2 */
  LED_3 = (1<<3)  /*!< Bit3 of port for LED3 */
} LED_Set;

#define setReg8Bits(RegName, SetMask) (RegName |= (byte)(SetMask))
#define LED_REG_DATA  PTFD

void LED_On(LED_Set Leds) {
  setReg8Bits(LED_REG_DATA, Leds);
}

void Test(void) {
  LED_On(LED_0|LED_1|LED_2|LED_3);
}
```

```
0000 87   [2] PSHA
0001 b600 [3] LDA _PTFD
0003 95   [2] TSX
0004 fa   [3] ORA ,X
0005 b700 [3] STA _PTFD
0007 8a   [3] PULH
0008 81   [6] RTS
```

```
000c a60f [2] LDA #15
000e 2000 [3] BRA LED_On
```

13 Bytes Code, 27 Cycles

LED's (inlined)

```
/* led.h */
#define LED_On(leds) (setReg8Bits(LED_REG_DATA, leds))
```

```
void Test(void) {
  LED_On(LED_0|LED_1|LED_2|LED_3);
}
```

```
0000 b600 [3] LDA _PTFD
0002 a4f0 [2] AND #-16
0004 b700 [3] STA _PTFD
0006 81   [6] RTS
```

7 Bytes Code,
14 Cycles

vs.

Function Call

```
0000 87   [2] PSHA
0001 b600 [3] LDA _PTFD
0003 95   [2] TSX
0004 fa   [3] ORA ,X
0005 b700 [3] STA _PTFD
0007 8a   [3] PULH
0008 81   [6] RTS
```

```
000c a60f [2] LDA #15
000e 2000 [3] BRA LED_On
```

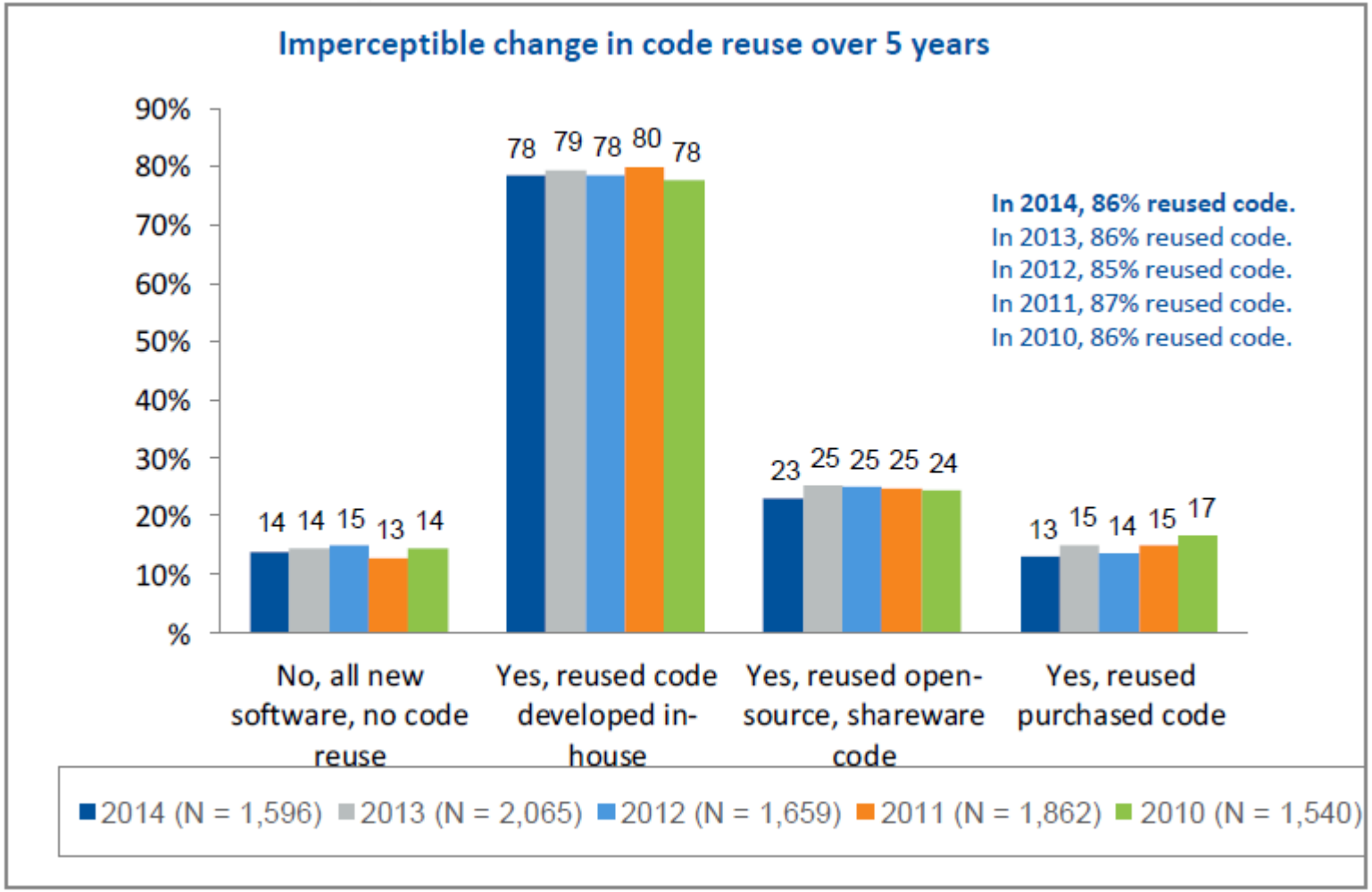
13 Bytes Code,
27 Cycles

LED Macros

```
#define LED_REG_DATA   PTFD
#define LED_USE_MACROS 1
#define LED_On(leds)   (setReg8Bits(LED_REG_DATA, leds))
```

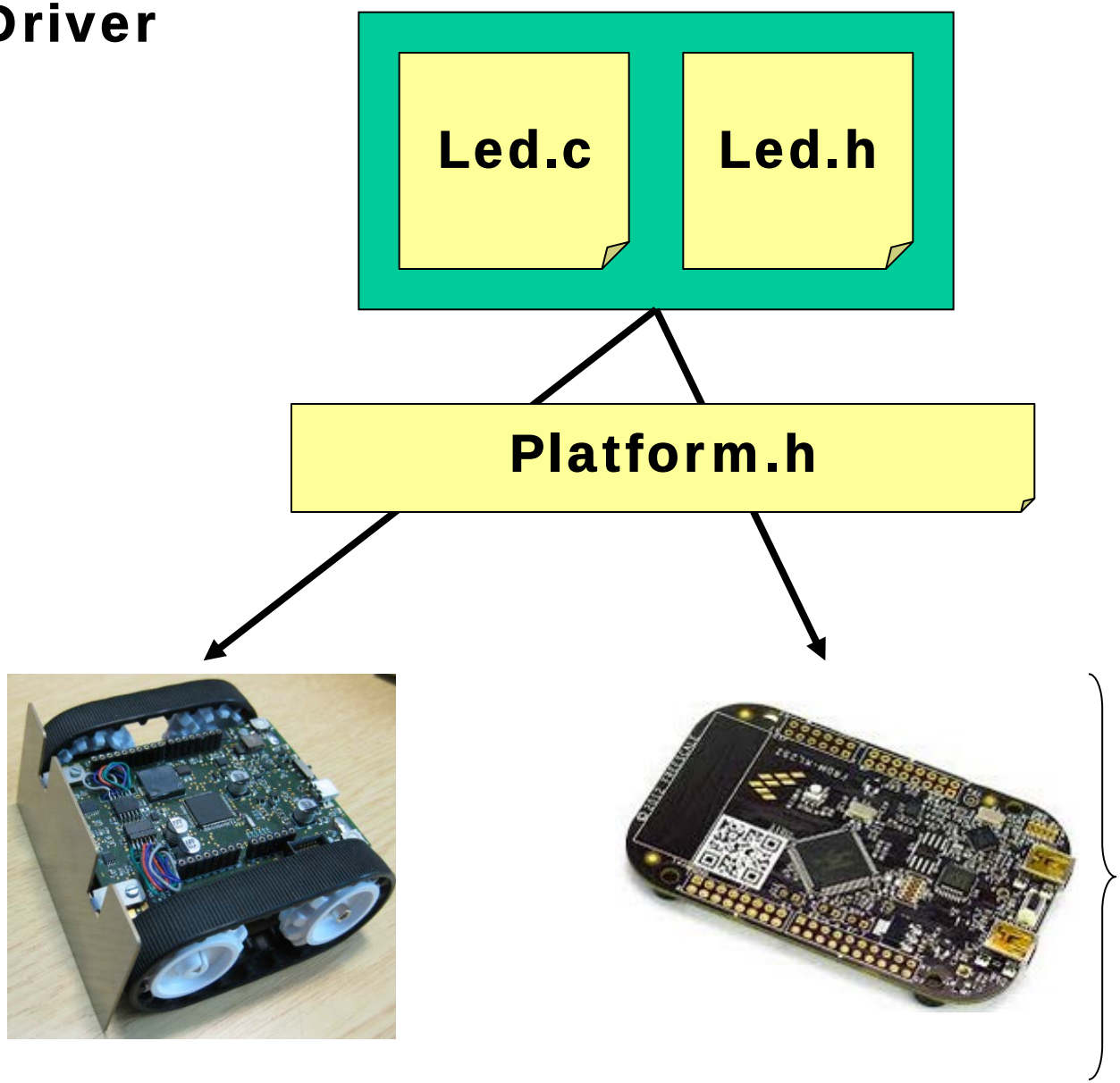
- Advantages
  - Faster code
  - Smaller Code
- Disadvantages
  - Interface
  - Encapsulation
  - Debugging
- Compromise
  - Both (Functions + Macros), optional
  - BUT: increased efforts for maintenance

Reusing Code



Source: TechInsight 2014 Embedded Market Study

LED Driver



Platform.h Configuration

```
/* Platform.h */
…

#include "derivative.h"

/* Macros which define our platform capabilities. Set to 0 if not supported. */
#define PL_HAS_LED      (1)   /*!< Set to 1 if platform has LEDs */

…
```

Derivative Interface

Platform Configuration

```
/* Platform.c */
#include "platform.h"

void PL_Init(void) {
#if PL_HAS_LED
  LED_Init();
#endif
}
```

Platform Target Board

```
/* platform.h */

/* list of hardware platforms supported */
#define K22FXROBO       1 /*!< K22FX512 Robot */
#define KL25ZFRDM       2 /*!< FRDM KL25Z128 */

#define PL_TARGET_BOARD KL25ZFRDM /*!< Sets the currently used platform */
```

```
/* led.h */
#include "platform.h"

#if PL_TARGET_BOARD==K22FXROBO
  #define PL_NOF_LED 2
#elif PL_TARGET_BOARD== KL25ZFRDM
  #define PL_NOF_LED 3
#else
  #error "unsupported target!"
#endif
```

Alternatively:

a) Eclipse build targets

b) -D compiler option

c) PEx Configuration

Macros for Configuration

```
/* platform.h */
#define PL_LED_USE_MACROS  1
```

```
/* led.h */
#include "platform.h"

#if PL_LED_USE_MACROS
  #define LED_On(leds)      (setReg8Bits(LED_REG_DATA, leds))
#else /* use function call */
  void LED_On(LED_Set Leds);
#endif
```

```
void Test(void) {
  LED_On(LED_0|LED_1|LED_2|LED_3);
}
```

Platform.h

- #defines control functionality
- Way to share implementation files

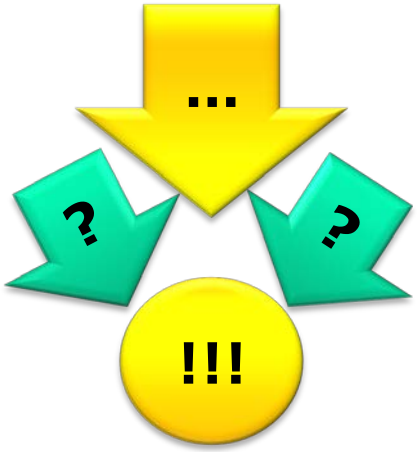
```
#define PL_BOARD_KIND_ROBO   1
  /*!< MK22 Robo board */
#define PL_BOARD_KIND_FRDM   2
  /*!< FRDM KL25Z board */
#if defined(__ROBO_BOARD__) /* compiler define!*/
  #define PL_BOARD   PL_BOARD_KIND_ROBO
  /*!< We are the Robo board */
#else
  #define PL_BOARD   PL_BOARD_KIND_FRDM
  /*!< We are the FRDM board */
#endif

#define PL_HAS_LED       0
#define PL_HAS_MOTOR   ( 1 && PL_BOARD==PL_BOARD_KIND_ROBO)
```

Summary

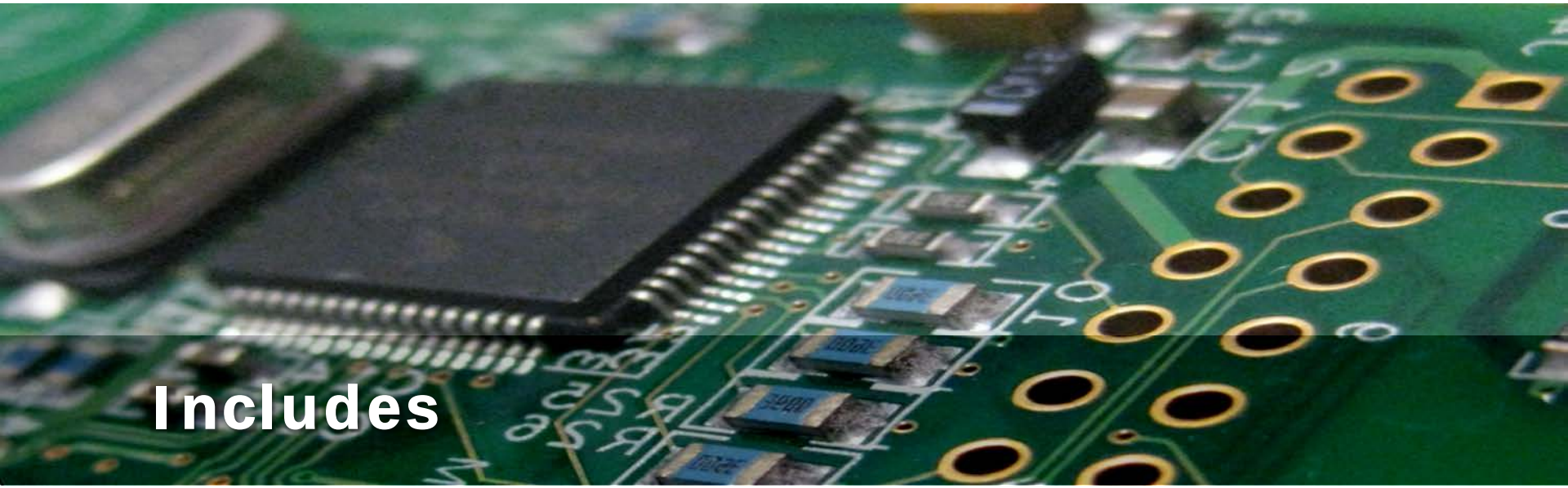
- *Problem: Efficient Implementation with C (LED)*

- Macros
  - What they are
  - Advantages
  - Disadvantages
  - Traps & Pitfalls (Parenthesis!)
  - Application hints

- Inlining (efficiency!)
- Configuration



Lucerne University of
Applied Sciences and Arts
HOCHSCHULE
LUZERN
Technik & Architektur



Includes

"I thought I did the right thing?…"

Prof. Erich Styger
erich.styger@hslu.ch
+41 41 349 33 01

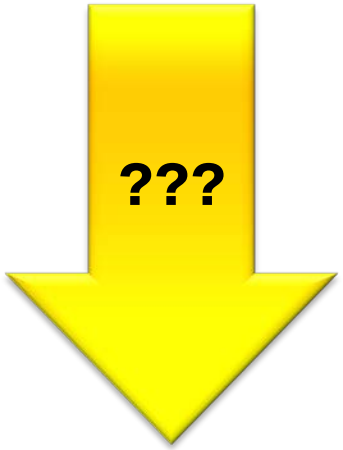
Learning Goals

- Goal
  - Understanding Compiler Includes

- Header File for Interface/Declaration
- Source File for Implementation

- Mechanics of #include
- Guarding
- What and Where

- Common Rules



Header / Source, what is where?

- Declaration: Name
- Definition: Memory allocation
- Convention:
  - *.c: Implementation File, Definition
  - *.h: Interface/Header File: external Declarations

```c
/* drv.c */
#include "drv.h"

int DRV_global = 7;
static int v;

void DRV_Init(void) {
  v = 3;
  DRV_global += v;
}
```

```c
/* drv.h */

#ifndef __DRV_H_
#define __DRV_H_

extern int DRV_global;

void DRV_Init(void);

#endif /* __DRV_H_ */
```

```c
/* main.c */

#include "drv.h"

void main(void) {
  DRV_Init();
  DRV_global++;
}
```

#include Directive

- Textual inclusion of files
- Result is ‘compilation unit’

```
/* b.h */
extern int b;
```

```
/* a.h */
extern int a;
```

```
/* m.h */
#include “b.h”
```

```
/* m.c */
#include “m.h”
#include “a.h”
void foo(void) {}
```

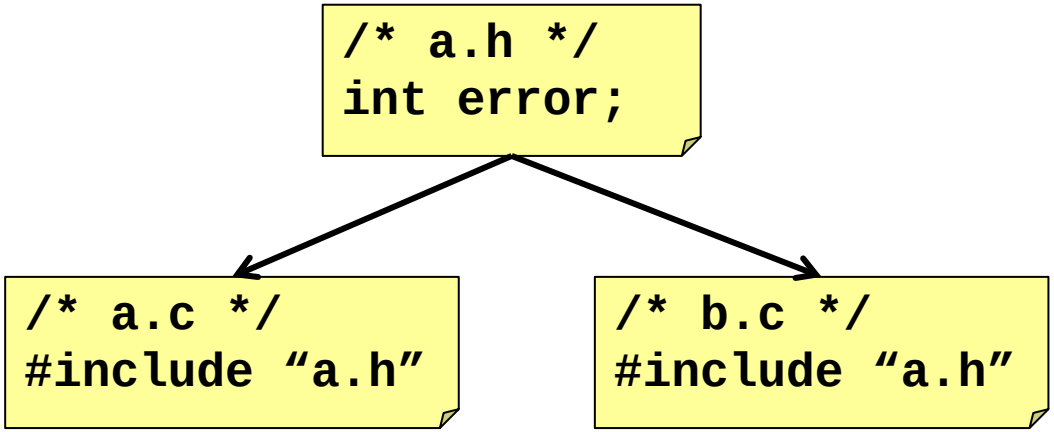
Preprocessor

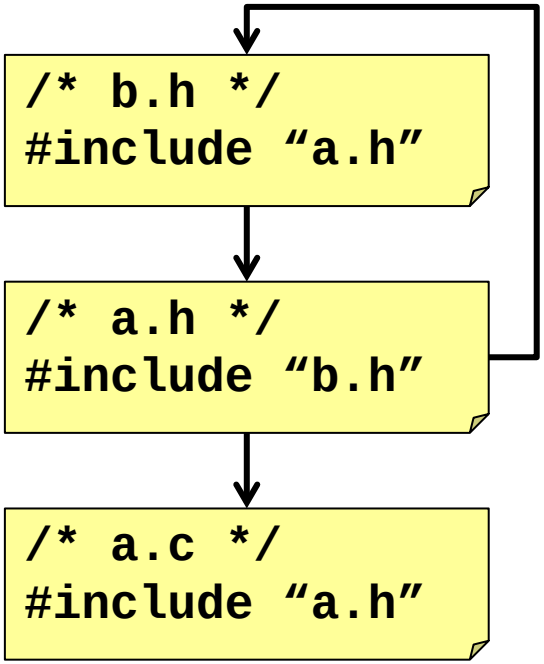
Compilation Unit

```
/* m.c */
/* m.h */
/* b.h */
extern int b;
/* a.h */
extern int a;
void foo(void) {}
```

#ifndef - #define - #endif

- Protection against
  - multiple declarations/definitions
  - Recursive includes





#ifndef - #define - #endif

- ‘Protection’ symbol
  - Avoid name conflicts!
  - Convention: __<*FileName*>_H_
  - Double Underscore: ‘reserved names’

```
/* platform.h */
#ifndef __PLATFORM_H_
#define __PLATFORM_H_
  #define PL_HAS_LED (1)
#endif /* __PLATFORM_H_ */
```

```
/* led.h */
#ifndef __LED_H_
#define __LED_H_
  void LED_On(void);
#endif /* __LED_H_ */
```

```
#include “platform.h”
#include “led.h”
void foo(void) {
  LED_On();
}
```

What and Where? Self-Containment

- As a developer, you could do whatever you want. BUT:…
- Include your own interface too!
- Header file should be ‘self contained’
  - Users of the interface should only need to include that interface
  - Using an interface/header file shall not depend on include order

```
/* led.c */
…
```

```
/* led.h */
#include “BitIO.h”
#define LED_On() \
   BitIO_BitOn()
```

```
/* main.c */
#include “LED.h”
void main(void) {
  LED_On();
}
```

```
/* led.c */
#include “BitIO.h”
include “LED.h”
void LED_On(void) {
  BitIO_BitOn();
}
```

```
/* led.h */
void LED_On(void);
```

```
/* main.c */
#include “LED.h”
void main(void) {
  LED_On();
}
```

What not to do…

- 'Reducing' includes in the wrong place is a bad thing

```
/* pwm.h */
…
```

```
/* led.h */
#include “led.h”
#include “pwm.h”
```

```
/* pwm.c */
…
```

```
/* led.c */
…
```

```
/* main.c */
#include “led.h”
void main(void) {
  LED_On();
  PWM_Off();
}
```

```
/* led.h */
…
```

```
/* pwm.h */
…
```

```
/* led.c */
…
```

```
/* pwm.c */
…
```

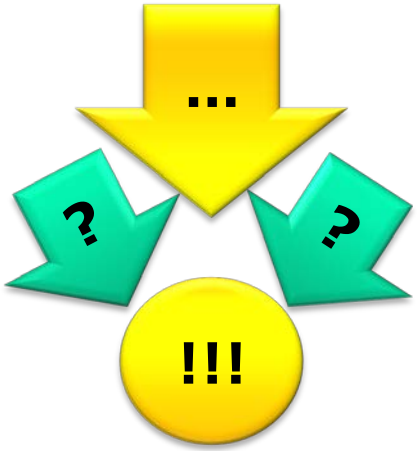
```
/* main.c */
#include “led.h”
#include “pwm.h”
void main(void) {
  LED_On();
  PWM_Off();
}
```

Summary with Rules

- Information hiding
  - Only expose in interface/header file what is needed
  - Not more, not less

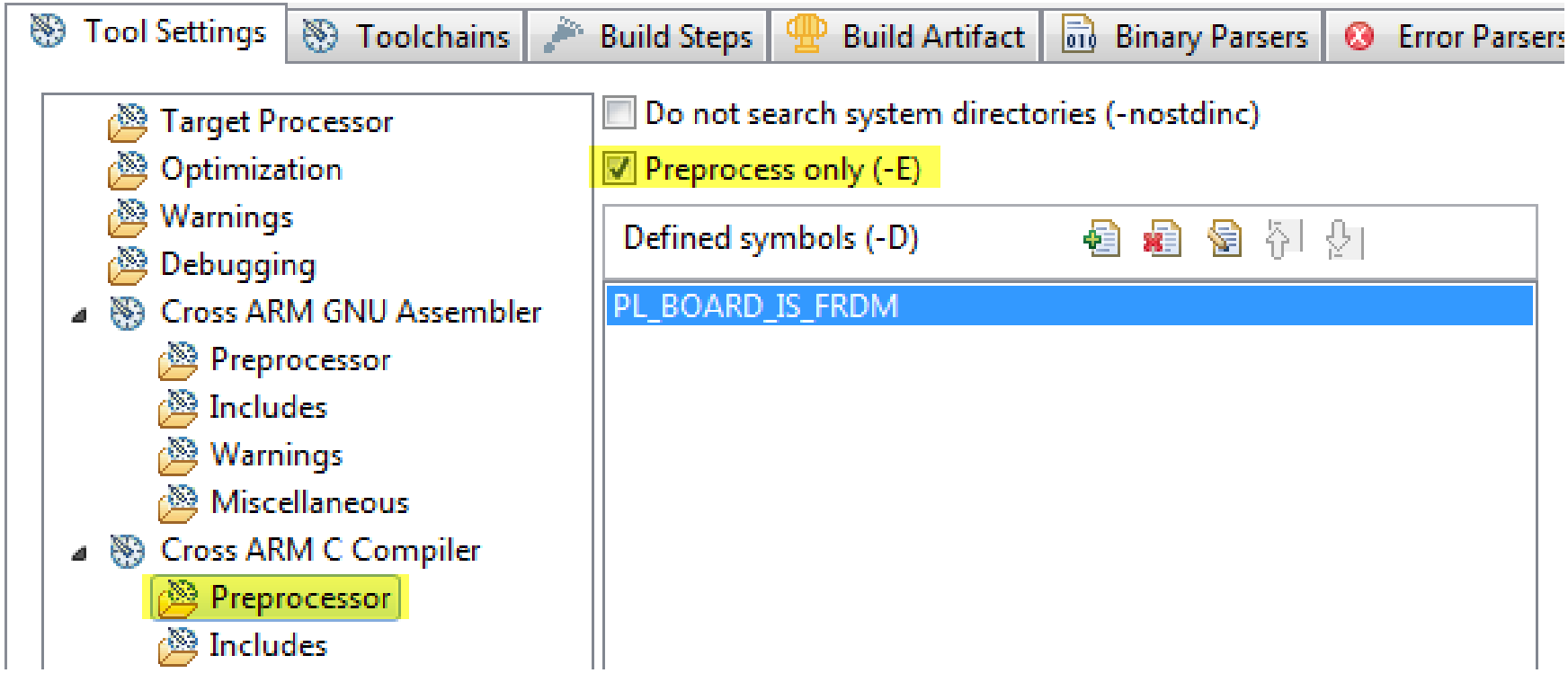
- Guard Header Files with #ifndef - #define - #endif
  - Use a guard define name with low chance of conflict

- Header file shall include only what is needed in the header file itself
- Source file includes the interfaces which are used for the implementation



Generating Preprocessor Output

- Enable –E
- Compiler does not produce object files!
  - *.o are preprocessor output (text files)



Preprocessor Output

- Extension *.o given by build system
- Open as text files
- Link phase will fail

main.o

```
# 1 "../Sources/main.c"
# 1 "C:\\Users\\tastyger\\Data\\HSLU\\Vorlesung\\INTRO_HS2014\\git\\INTRO_HS2014\\Proje
#define __STDC__ 1
# 1 "../Sources/main.c"
#define __STDC_VERSION__ 199901L
# 1 "../Sources/main.c"
#define __STDC_HOSTED__ 1
# 1 "../Sources/main.c"
#define __GNUC__ 4
# 1 "../Sources/main.c"
#define __GNUC_MINOR__ 8
# 1 "../Sources/main.c"
#define __GNUC_PATCHLEVEL__ 0
# 1 "../Sources/main.c"
#define __VERSION__ "4.8.0"
# 1 "../Sources/main.c"
#define __ATOMIC_RELAXED 0
# 1 "../Sources/main.c"
```

Lab 8: Preprocessor (15")

- *Problem: Macro Debugging*

- Generate Preprocessor Listing
- Inspect Preprocessing Listing