

Creating Software Interfaces for Hardware

Embedded Software Essentials

Platform Independence [S1a]

- Microcontrollers have a variety of

- Peripherals
- Registers
- Memories

- **Software Architecture:** structured organization of a software project

- Design software to be

independent of architecture
¹Impossible to remove all hardware dependencies and platform¹

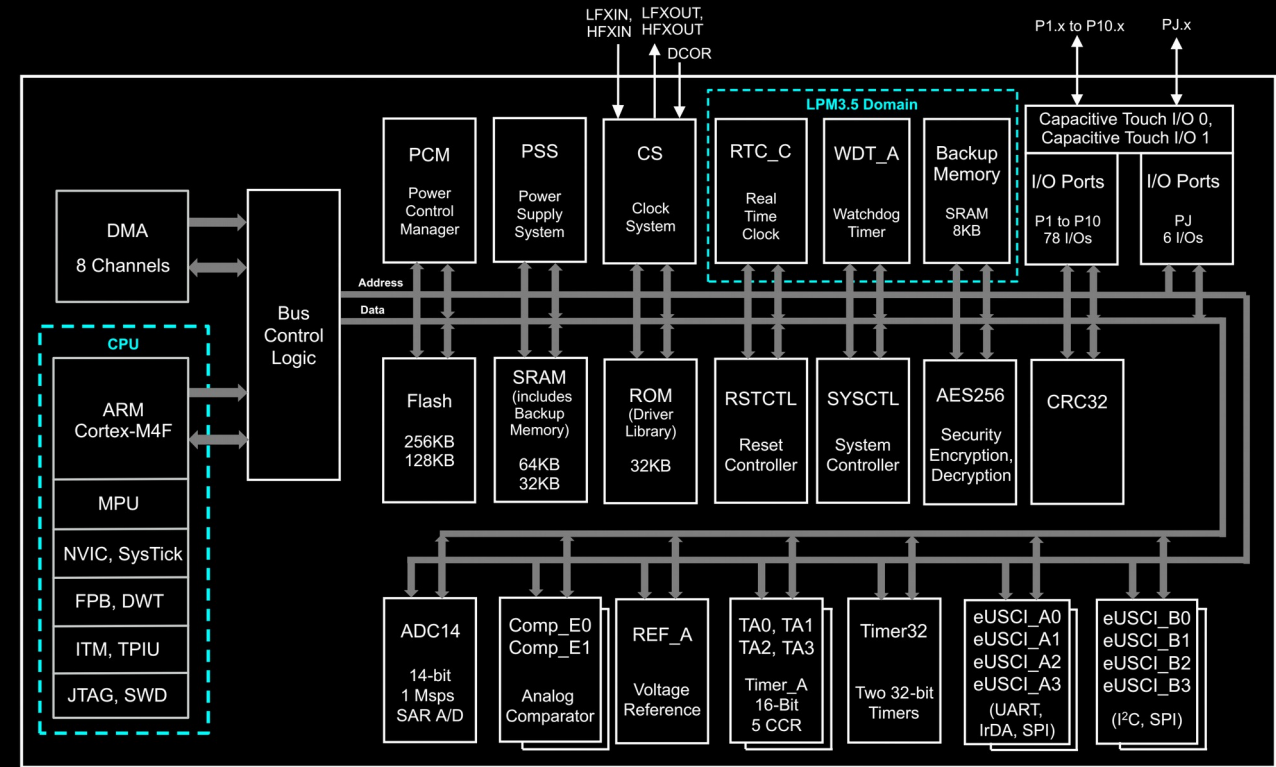


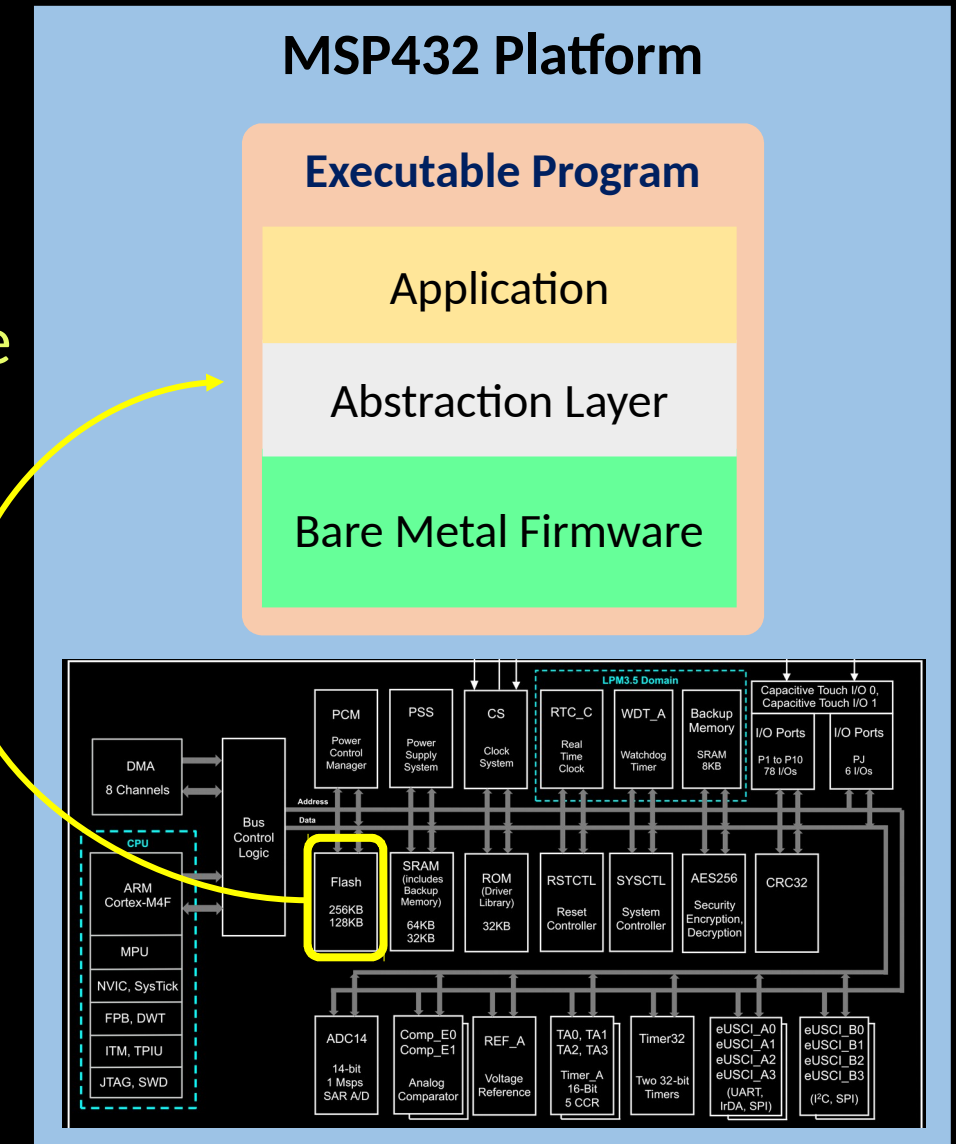
Figure 1-1. MSP432P401x Functional Block Diagram

May have 1000+ peripheral registers.

Platform Independence [S1b]

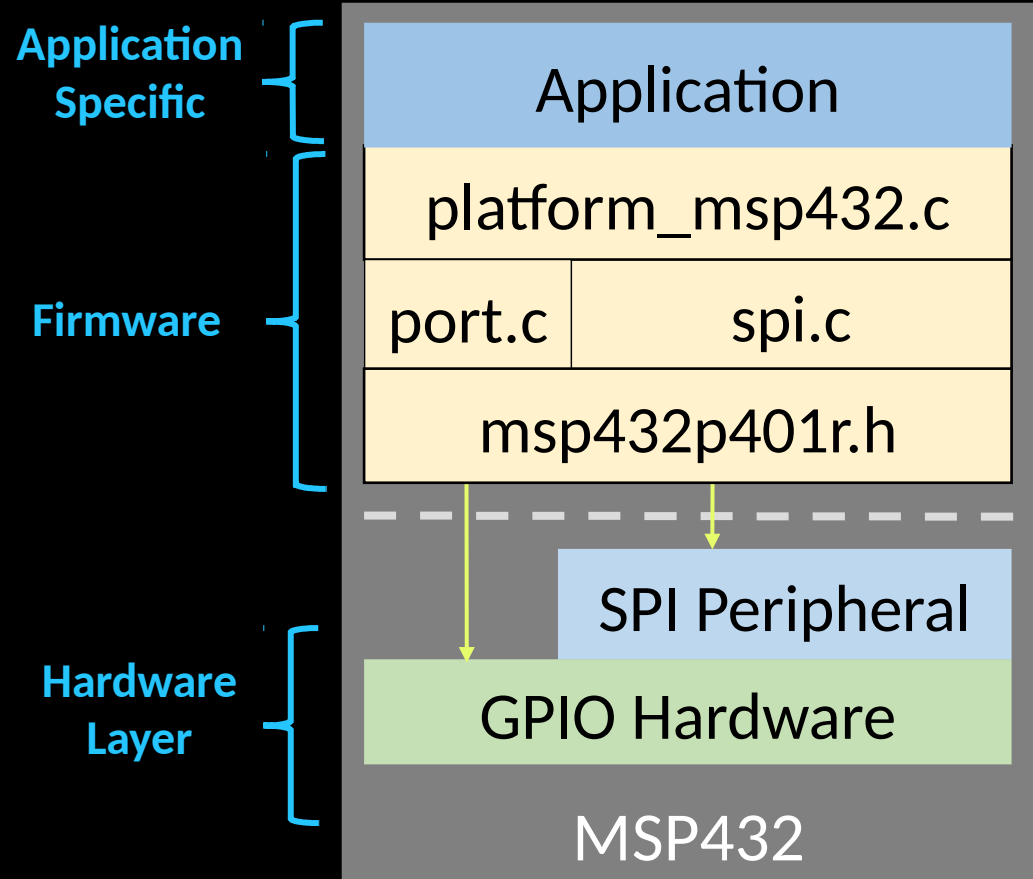
- Microcontrollers have a variety of
 - Peripherals
 - Registers
 - Memories
- **Software Architecture:** structured organization of a software project
- Design software to be **independent** of architecture and platform¹

Executable
stored in
Flash



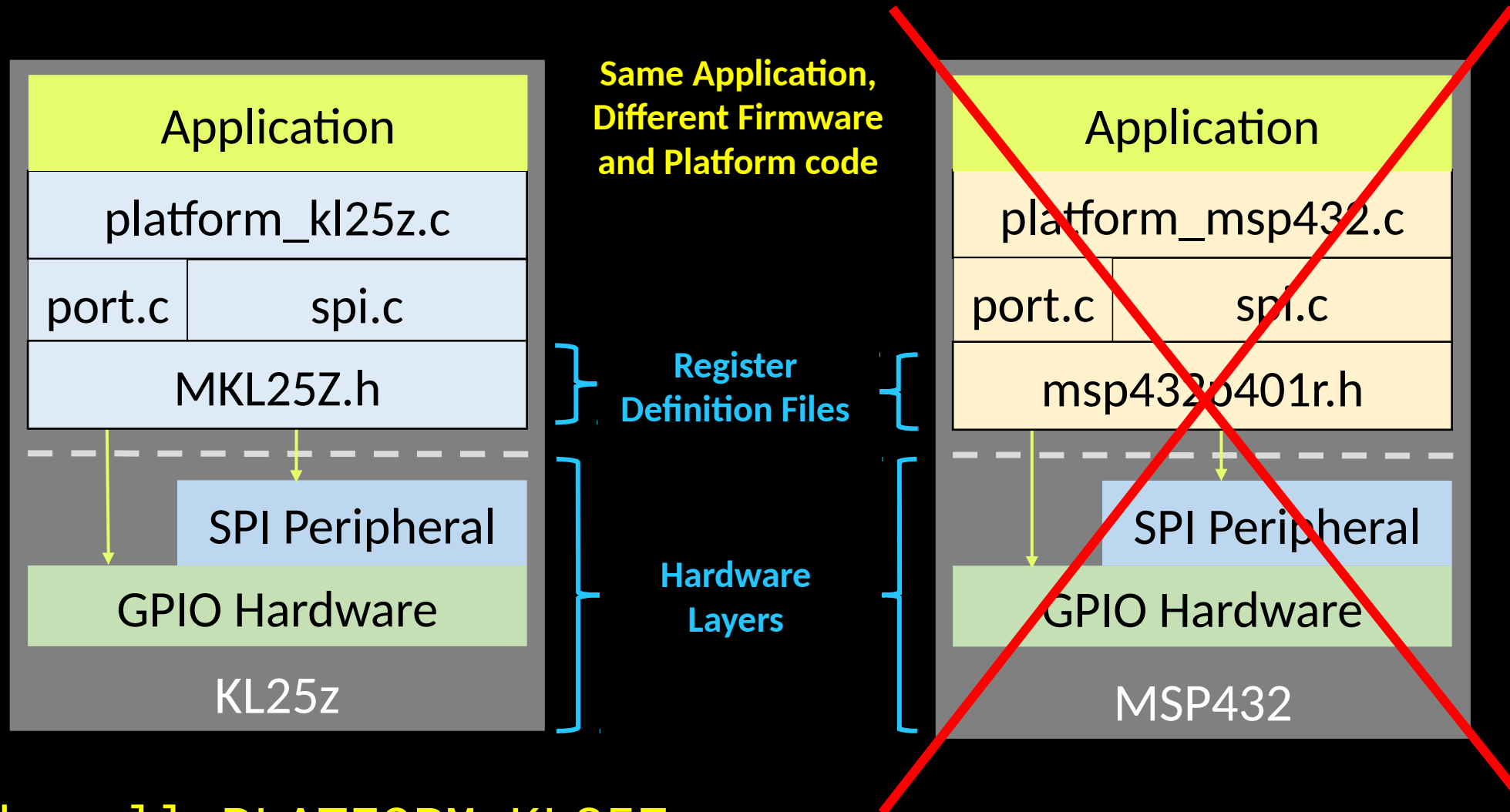
Abstraction Layers [S2a]

- Firmware layer needs to be efficient and bug-free
- Make higher level software **independent** of low level firmware
 - Example: application software



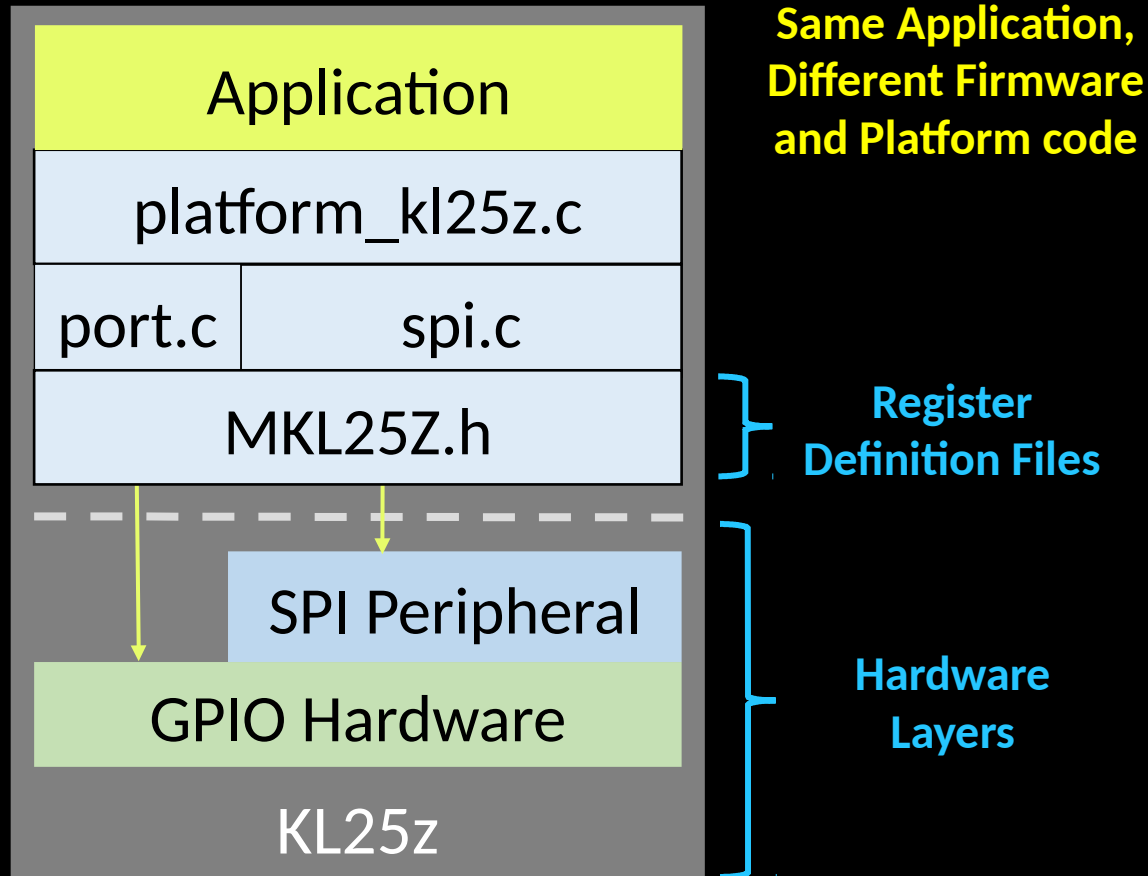
\$ make all PLATFORM=MSP432

Abstraction Layers [S2b]



\$ make all PLATFORM=KL25Z

Abstraction Layers [S2c]



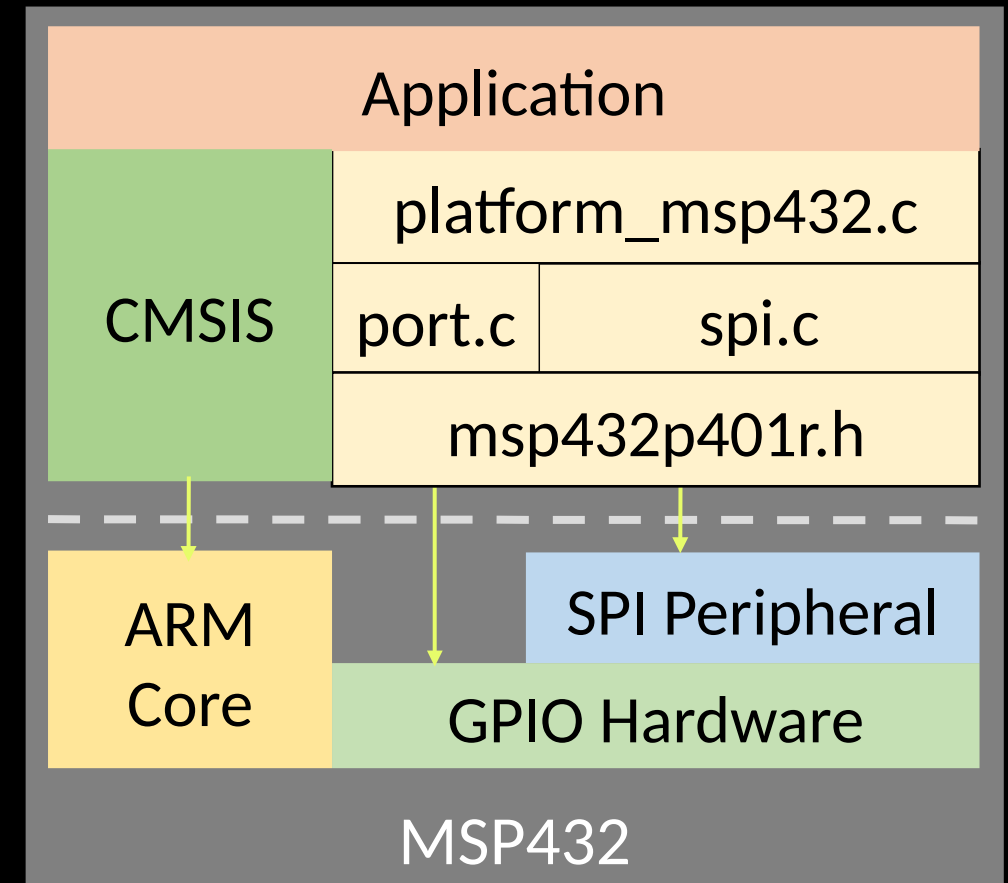
- Interface methods for hardware
 - Register Definition Files
 - Macro Functions
 - Specialized C-Functions

```
$ make all PLATFORM=KL25Z
```

Embedded Hardware Interface

[S4a]

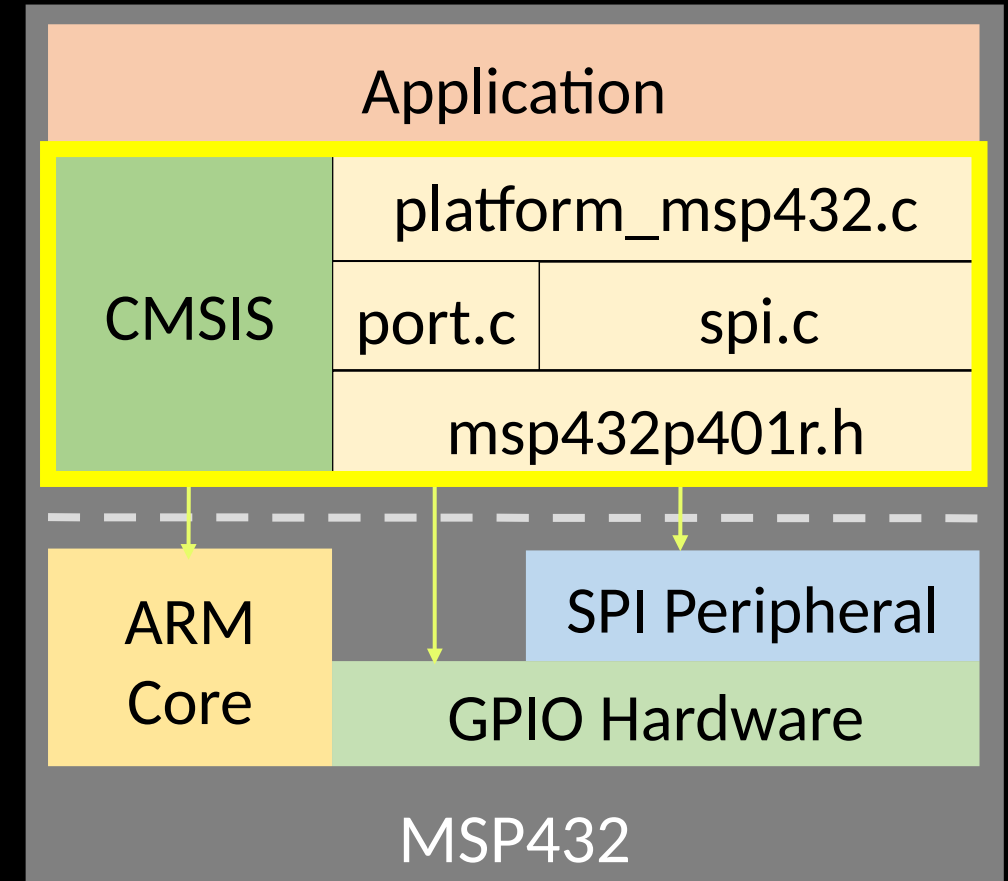
- Interface methods for hardware
 - Register Definition Files
 - Macro Functions
 - Specialized C-Functions



Embedded Hardware Interface

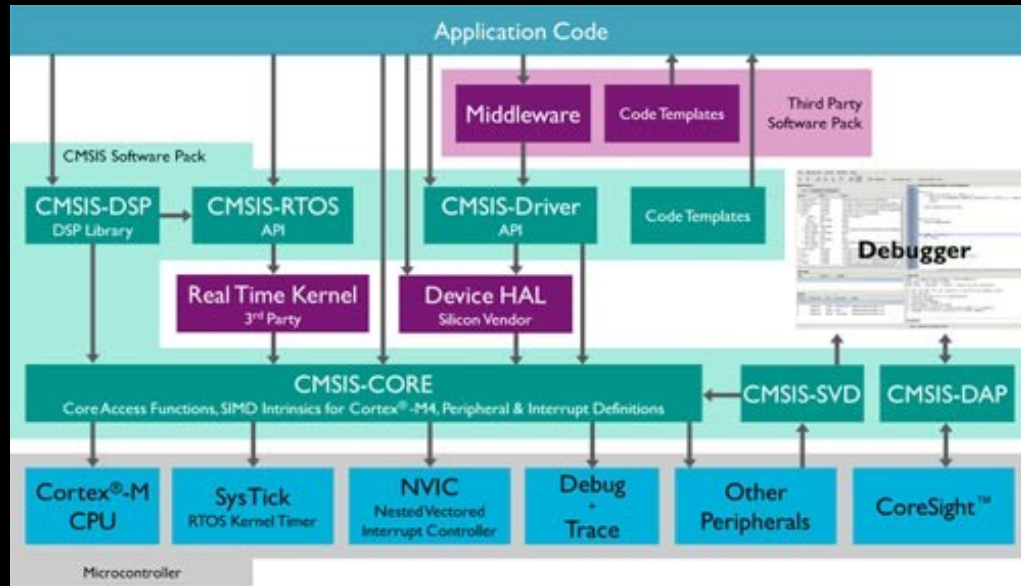
[S4b]

- Interface methods for hardware
 - Register Definition Files
 - Macro Functions
 - Specialized C-Functions
- Create **Hardware Abstraction Layer (HAL)** to simplify application level programming
 - Acts like an API (Application Programming Interface)

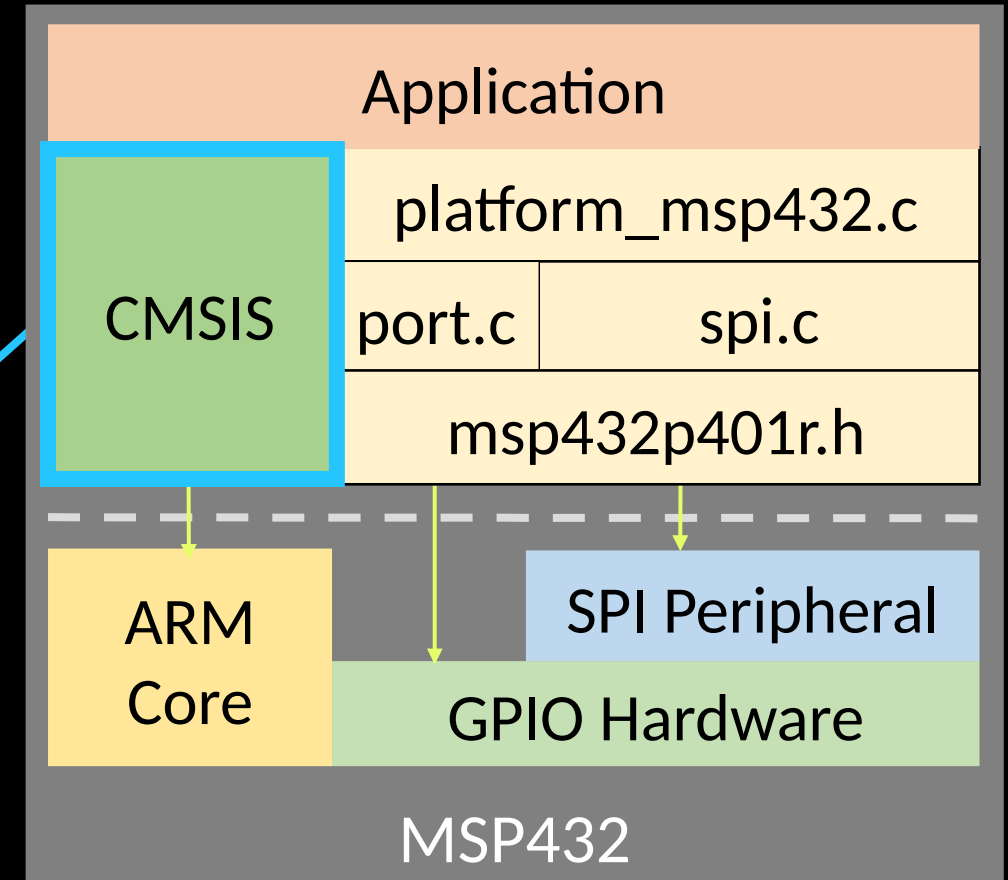


Embedded Hardware Interface [S4c]

- Create **Hardware Abstraction Layer (HAL)** to simplify application level programming



ARM Core Microcontroller Software Interface Standard (CMSIS) Diagram



Register Definition Files [S5a]

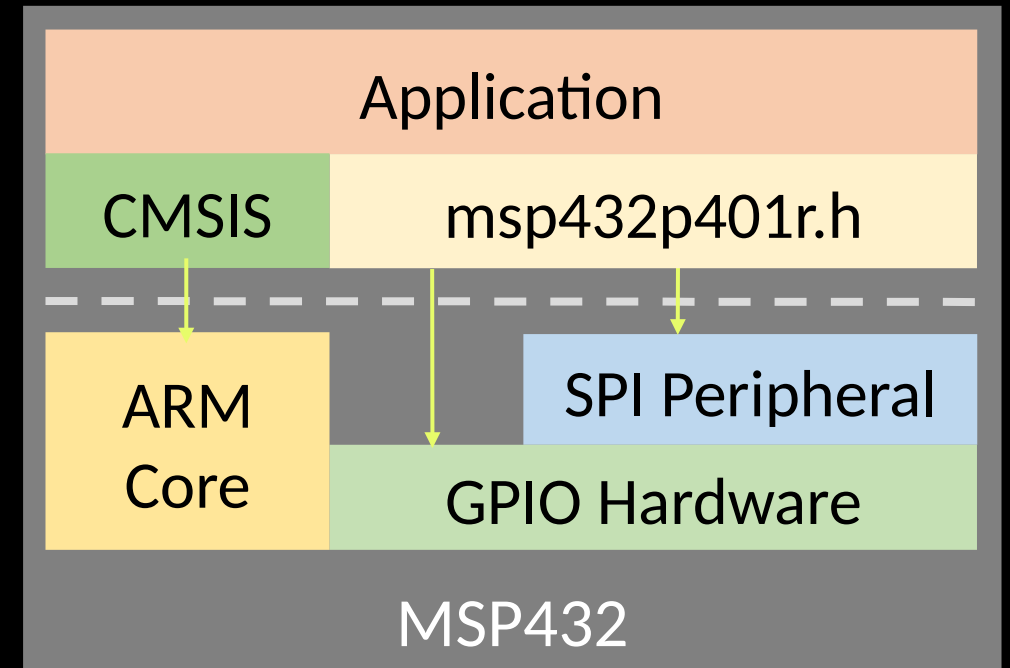
- Platform File that provides interface to peripheral memory by specifying
 - **Address List** for Peripherals
 - **Access Methods**
 - Defines for **Bit Fields** and **Bit Masks**

Private Peripherals	General Peripherals
Processor	ADC14
Debug Control	Reserved
SCB	Timer32
FPU	Port
	WDT_A
	RTC_C
MPU	CRC32
NVIC	AES256
Reserved	COMP_E0-E1
SysTick Timer	REF_A
Misc system control registers	eUSCI_B0-B3
	eUSCI_A0-A3
	Timer_A0-A3

Could Contain 1000's of Registers!

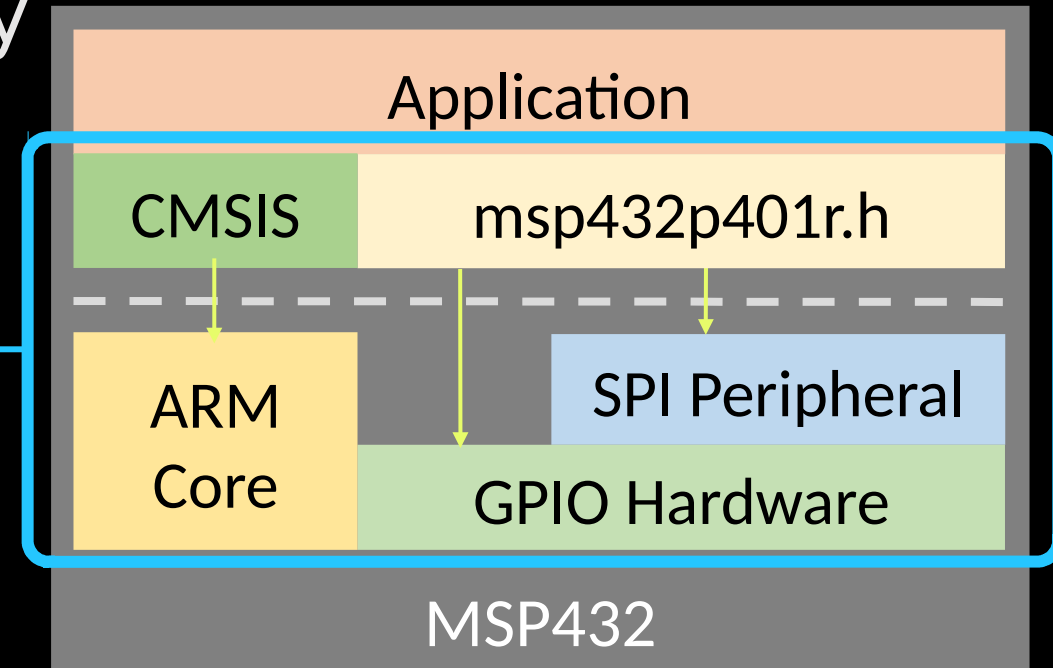
Register Definition Files [S5b]

- Platform File that provides interface to peripheral memory by specifying
 - **Address List** for Peripherals
 - **Access Methods**
 - Defines for **Bit Fields** and **Bit Masks**
- Peripheral **Access Methods** used to read/write data
 - Preprocessor Macros
 - Direct Dereferencing of Memory
 - Structure Overlays



Register Definition Files [S5c]

- Platform File that provides interface to peripheral memory by specifying
 - **Address List** for Peripherals
 - **Access Methods**
 - Defines for **Bit Fields** and **Bit Masks**
- Peripheral **Access Methods** used to read/write data
 - Preprocessor Macros
 - Direct Dereferencing of Memory
 - Structure Overlays



Platform and
architecture software
interface

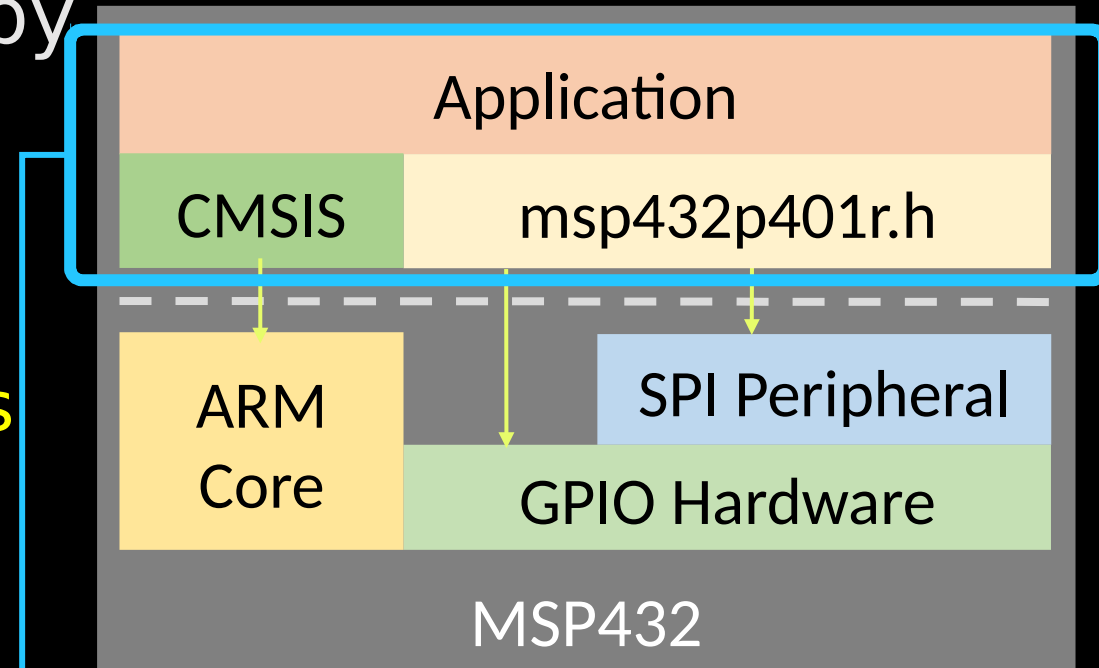
Register Definition Files [S5d]

- Platform File that provides interface to peripheral memory by specifying

- **Address List** for Peripherals
- **Access Methods**
- Defines for **Bit Fields** and **Bit Masks**

- Peripheral **Access Methods** used to read/write data

- Preprocessor Macros
- Direct Dereferencing of Memory
- Structure Overlays



Direct Application
dependency on
platform and core
architecture

Embedded Hardware Interface

[S6a]

- Interface methods for hardware
 - Register Definition Files
 - Macro Functions
 - Specialized C-Functions

Embedded Hardware Interface

[S6b]

- Interface methods for hardware

- Register Definition Files

- Macro Functions

- Specialized C-Functions

} Regularly used in
firmware abstraction

Embedded Hardware Interface [S6c]

- Interface methods for hardware
 - Register Definition Files
 - Macro Functions
 - Specialized C-Functions
- Regularly used in firmware abstraction

/* Macro Function to Read Memory */

```
#define HWREG16(x) (*((volatile uint16_t *) (x)))
```

—————> Platform Dependency

Embedded Hardware Interface

[S6d]

- Interface methods for hardware

- Register Definition Files

- Macro Functions

- Specialized C-Functions

} Regularly used in
firmware abstraction

```
/* Macro Function to Read Memory */
```

```
#define HWREG16(x) (*((volatile uint16_t *)(x)))
```

—————> Platform Dependency

```
/* Specialized CMSIS Function to Enable Interrupts */
```

```
__attribute__((always_inline)) __STATIC_INLINE void
```

```
__enable_irq(void)
```

```
{
```

—————> Architecture Dependency

```
    __ASM volatile ("cpsie i" ::: "memory");
```

```
}
```

Macro Functions [S7a]

- Direct Substitution in code

```
#define TA0CTL_ADDR (0x40000000)
/* 16 Bit Register Access Macros */
#define HWREG16(x) (*((volatile uint16_t *)(x)))
```

Macro Functions [S7b]

- Direct Substitution in code

```
#define TA0CTL_ADDR (0x40000000)
```

```
/* 16 Bit Register Access Macros */
```

```
#define HWREG16(x) (*((volatile uint16_t *) (x)))
```

Not an actual function call, no calling
convention overhead!

Macro Functions [S7c]

- Direct Substitution in code

```
#define TA0CTL_ADDR (0x40000000)
```

```
/* 16 Bit Register Access Macros */
```

```
#define HWREG16(x) (*((volatile uint16_t *) (x)))
```

- Example use of access method:

```
HWREG16(TA0CTL_ADDR) = 0x0202;
```

```
((*(volatile uint16_t *) (0x40000000))) = 0x0202;
```

Macro Functions [S7d]

/ 8, 16, & 32 Bit Register Access Macros */*

```
#define HWREG8(x)    (*((volatile uint8_t *) (x)))
```

```
#define HWREG16(x)   (*((volatile uint16_t *) (x)))
```

```
#define HWREG32(x)   (*((volatile uint32_t *) (x)))
```

```
#define TA0CTL        (HWREG16(0x40000000))
```

Example Use of Access Macro:

```
TA0CTL = 0x0202;
```



```
volatile uint16_t * ta0_ctrl = (uint16_t *) 0x40000000;  
*ta0_ctrl = 0x0202;
```

Macro Functions [S7e]

/ 8, 16, & 32 Bit Register Access Macros */*

```
#define HWREG8(x)    (*((volatile uint8_t *) (x)))
```

```
#define HWREG16(x)   (*((volatile uint16_t *) (x)))
```

```
#define HWREG32(x)   (*((volatile uint32_t *) (x)))
```



Pass in any address
to access a register



```
#define TA0CTL        (HWREG16(0x40000000))
```

Example Use of Access Macro:

```
TA0CTL = 0x0202;
```



Same

```
volatile uint16_t * ta0_ctrl = (uint16_t *) 0x40000000;
```

```
*ta0_ctrl = 0x0202;
```

Port Macros [S8a]

- Generic memory access macro for different sized registers
 - 8, 16, 32 Bit Registers

```
/* 8, 16, & 32 Bit Register Access Macros */  
#define HWREG8(x)      (*((volatile uint8_t *) (x)))  
#define HWREG16(x)     (*((volatile uint16_t *) (x)))  
#define HWREG32(x)     (*((volatile uint32_t *) (x)))
```

```
/* Port 1 Register Access Macros */  
#define P1IN           (HWREG8(0x40004C00))  
#define P1OUT          (HWREG8(0x40004C02))  
#define P1DIR          (HWREG8(0x40004C04))  
#define P1SEL0         (HWREG8(0x40004C0A))  
#define P1SEL1         (HWREG8(0x40004C0C))  
#define P1IES          (HWREG8(0x40004C18))  
#define P1IE           (HWREG8(0x40004C1A))
```

Port Macros [S8b]

- Generic memory access macro for different sized registers
 - 8, 16, 32 Bit Registers

- Port registers use the generalized access macros

```
/* Set P1.0 to Output Direction */  
P1DIR |= 0x01;
```

```
/* 8, 16, & 32 Bit Register Access Macros */  
#define HWREG8(x)      (*((volatile uint8_t *) (x)))  
#define HWREG16(x)     (*((volatile uint16_t *) (x)))  
#define HWREG32(x)     (*((volatile uint32_t *) (x)))
```

```
/* Port 1 Register Access Macros */  
#define P1IN            (HWREG8(0x40004C00))  
#define P1OUT           (HWREG8(0x40004C02))  
#define P1DIR           (HWREG8(0x40004C04))  
#define P1SEL0          (HWREG8(0x40004C0A))  
#define P1SEL1          (HWREG8(0x40004C0C))  
#define P1IES           (HWREG8(0x40004C18))  
#define P1IE            (HWREG8(0x40004C1A))
```

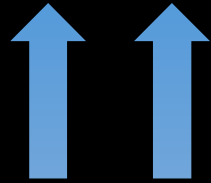

Port Macros [S8c]

```
#define SET_PORT_PIN_DIR (port, pin) ( (port)->DIR) |= (1 << pin) )
```

```
/* Set bit P1.0 to be output Direction */  
SET_PORT_PIN_DIR( P1, PIN0 );
```

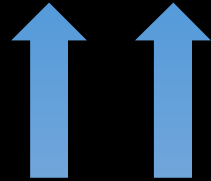
Port Macros [S8c]

```
#define SET_PORT_PIN_DIR (port, pin) ( (port)->DIR) |= (1 << pin) )
```



```
/* Set bit P1.0 to be output Direction */
```

```
SET_PORT_PIN_DIR( P1, PIN0 );
```



Port Macros [S8d]

```
#define SET_PORT_PIN_DIR (port, pin) ( (port)->DIR) |= (1 << pin) )
```

```
/* Set bit P1.0 to be output Direction */
```

```
SET_PORT_PIN_DIR( P1, PIN0 );
```

```
/* Define the PORT Constants and Types */
```

```
#define PIN0 (0x0)
```

```
#define DIO_PORT1_ADDR ((uint32_t) 0x40004C00)
```

```
#define P1 ( (DIO_PORT_Type*)(DIO_PORT1_ADDR) )
```

Port Structure Overlay

```
typedef struct {  
    __IO uint8_t IN;  
    uint8_t RESERVED0;  
    __IO uint8_t OUT;  
    uint8_t RESERVED1;  
    __IO uint8_t DIR;  
    uint8_t RESERVED2;  
    __IO uint8_t REN;  
    uint8_t RESERVED3;  
    ... /* More Registers */  
} DIO_PORT_TYPE;
```

Port Macros [S8e]

```
#define SET_PORT_PIN_DIR (port, pin) ( (port)->DIR) |= (1 << pin) )
```

```
/* Set bit P1.0 to be output Direction */
```

```
SET_PORT_PIN_DIR( P1, PIN0 );
```

```
/* Define the PORT Constants and Types */
```

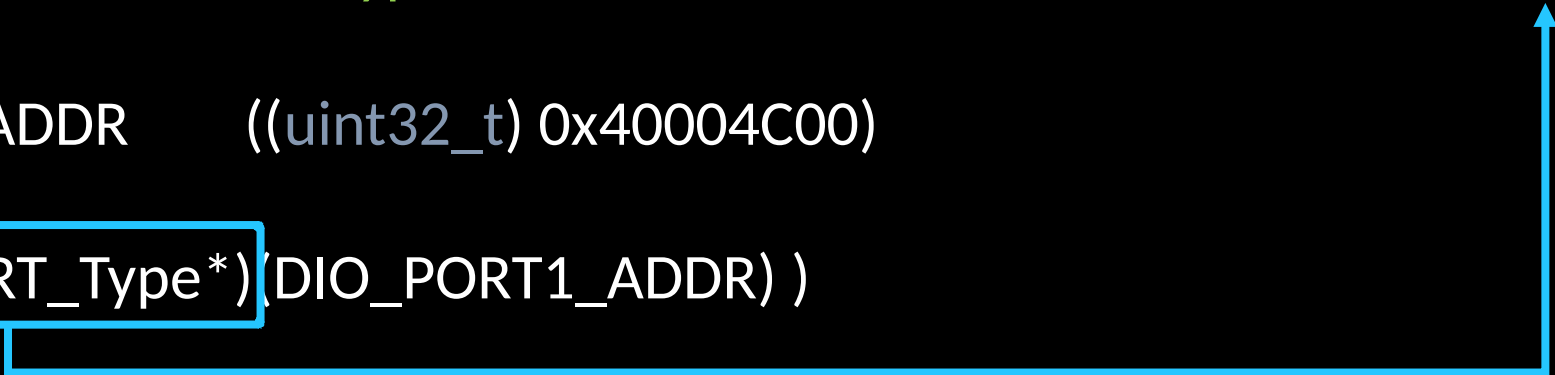
```
#define PIN0 (0x0)
```

```
#define DIO_PORT1_ADDR ((uint32_t) 0x40004C00)
```

```
#define P1 ( (DIO_PORT_Type*) (DIO_PORT1_ADDR) )
```

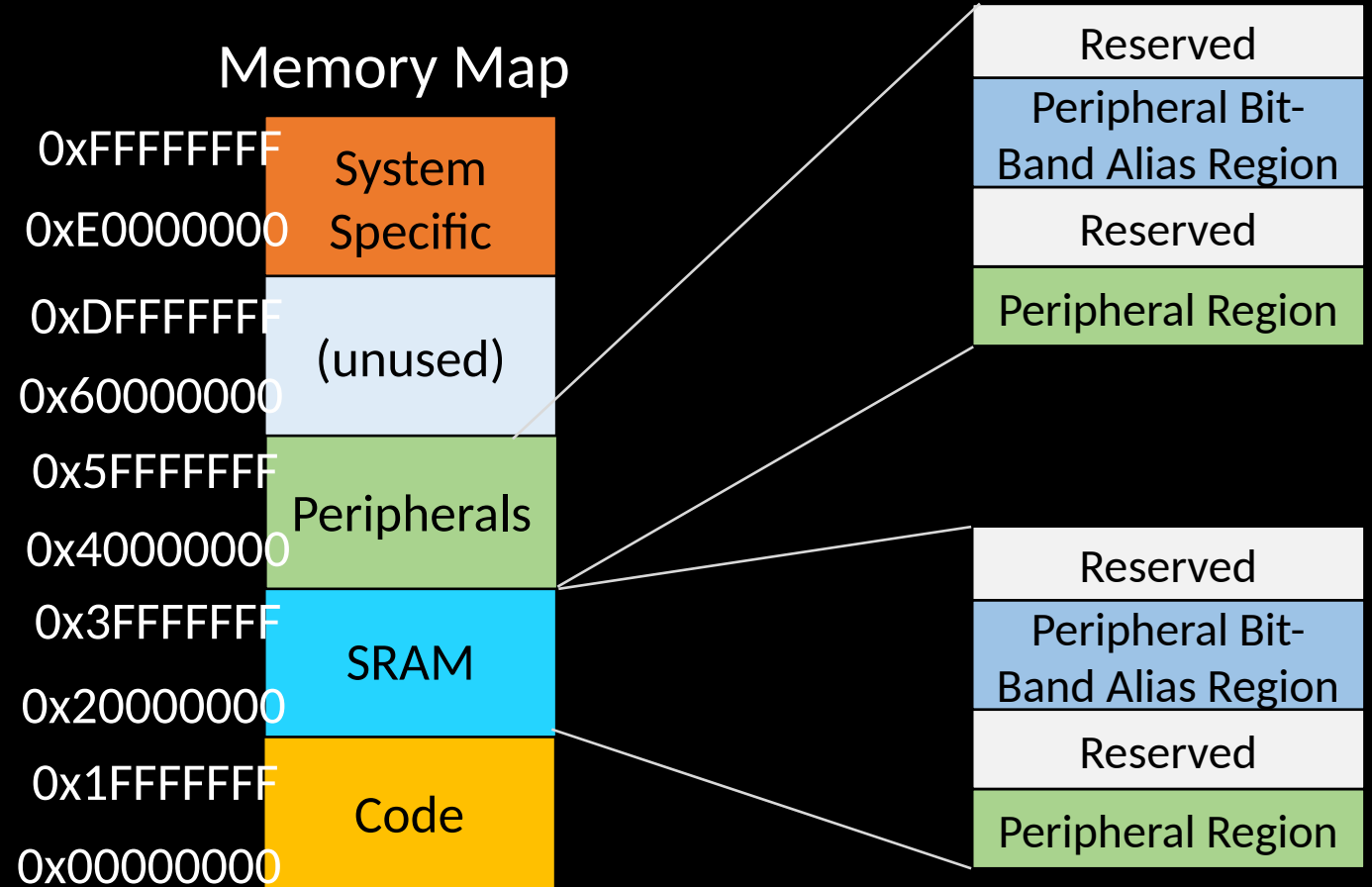
Port Structure Overlay

```
typedef struct {  
    __IO uint8_t IN;  
    uint8_t RESERVED0;  
    __IO uint8_t OUT;  
    uint8_t RESERVED1;  
    __IO uint8_t DIR;  
    uint8_t RESERVED2;  
    __IO uint8_t REN;  
    uint8_t RESERVED3;  
    ... /* More Registers */  
} DIO_PORT_Type;
```



Bit Band Macros [S9b]

- Bit Band Region allows for **atomic** reads/writes of single bits for first 1MB of SRAM and Peripheral Memory
- Not all of the SRAM & Peripheral Memory has a bit-band alias



Bit Band Macros [S9b]

```
#define TA0CTL_ADDR (0x40000000)
```

```
/* Bit Band Region is offset from Peripheral/SRAM */
```

```
#define BB_OFFSET (0x02000000)
```

```
/* Macro Function to Read Memory */
```

```
#define HWREG32(addr) (*((volatile uint32_t *) (addr)))
```

```
/* Bit Band Alias Offset Address */
```

```
#define BITBAND_ADDR(addr, bit) ( (addr & 0xF0000000) + BB_OFFSET + \  
                                ((addr & 0xFFFFF) << 5) + (bit << 2) )
```

```
/* Set bit 1 of TA0CTL Register */
```

```
HWREG32( BITBAND_ADDR(TA0CTL_ADDR, 1) );
```

Bit Band Macros [S9c]

```
#define TA0CTL_ADDR (0x40000000)
```

```
/* Bit Band Region is offset from Peripheral/SRAM */
```

```
#define BB_OFFSET (0x02000000)
```

```
/* Macro Function to Read Memory */
```

```
#define HWREG32(addr) (*((volatile uint32_t *) (addr)))
```

```
/* Bit Band Alias Offset Address */
```

```
#define BITBAND_ADDR(addr, bit) ( (addr & 0xF0000000) + BB_OFFSET + \  
                                ((addr & 0xFFFFF) << 5) + (bit << 2) )
```

```
/* Set bit 1 of TA0CTL Register */
```

```
HWREG32( BITBAND_ADDR(TA0CTL_ADDR, 1) );
```


Macro Problems [S10a]

- Numerous Issues with Macro Functions
 - No Type Checking
 - Bug Introduction
 - Complex / Confusing Layers of Macros Calling Macros
 - Code Size and Duplication

Macro Problems [S10b]

- Numerous Issues with Macro Functions
 - No Type Checking
 - Bug Introduction
 - Complex / Confusing Layers of Macros Calling Macros
 - Code Size and Duplication

```
#define PERIPH_BASE ((uint32_t) 0x40000000)
#define DIO_BASE     (PERIPH_BASE + 0x00004C00)
#define P1_BASE      (DIO_BASE + 0x0000)
#define P1DIR_ADDR   (P1BASE + 0x0004)
```



Must specify type explicitly

Before Preprocessing

```
#define SQUARE(x) ((x)*(x))
uint32_t y = 2;
uint32_t Y_sqrd = ;
Y_sqrd = SQUARE(y++);
```

After Preprocessing

```
uint32_t y = 2;
uint32_t Y_sqrd = ;
Y_sqrd = ((y++)*(y++));
```

Undefined Operation

Macro Problems [S10c]

- Numerous Issues with Macro Functions
 - No Type Checking
 - Bug Introduction
 - Complex / Confusing Layers of Macros Calling Macros
 - Code Size and Duplication

```
graph TD; A["#define PERIPH_BASE ((uint32_t) 0x40000000)"] --> B["#define DIO_BASE (PERIPH_BASE + 0x00004C00)"]; B --> C["#define P1_BASE (DIO_BASE + 0x0000)"]; C --> D["#define P1DIR_ADDR (P1BASE + 0x0004)"]; D --> E["#define HWREG8(x) (*((volatile uint8_t *)(x))]"]; D --> F["#define P1DIR (HWREG8(P1DIR_ADDR))"];
```

```
#define PERIPH_BASE ((uint32_t) 0x40000000)
#define DIO_BASE (PERIPH_BASE + 0x00004C00)
#define P1_BASE (DIO_BASE + 0x0000)
#define P1DIR_ADDR (P1BASE + 0x0004)
#define HWREG8(x) (*((volatile uint8_t *)(x)))
#define P1DIR (HWREG8(P1DIR_ADDR))
```

Macro Problems [S10d]


- Numerous Issues with Macro Functions
 - No Type Checking
 - Bug Introduction
 - Complex / Confusing Layers of Macros Calling Macros
 - Code Size and Duplication

The preprocessor will search and replace any used macro!

Specialized C Functions [S11]

- Functions are excellent for **encapsulating** a specialized operation
- Calling a traditional C-function can decrease performance due to calling convention overhead

Specialized C Functions [S11b]

- Functions are excellent for **encapsulating** a specialized operation
- Calling a traditional C-function can decrease performance due to calling convention overhead  **Need to reduce performance hit**

- Saving data to stack

- Creating local variables

- Branching

```
__attribute__((always_inline)) static inline void
```


```
__enable_irq(void)
```

```
{
```

```
    __ASM volatile ("cpsie i" ::: "memory");
```

```
}
```

Specialized C Functions [S11c]

- Functions are excellent for **encapsulating** a specialized operation
- Calling a traditional C-function can decrease performance due to calling convention overhead  **Need to reduce performance hit**

- Saving data to stack
- Creating local variables

- Branching

```
__attribute__((always_inline)) static inline void
```

```
__enable_irq(void)
```

```
{
```

```
    __ASM volatile ("cpsie i" ::: "memory");
```

```
}
```

Inline Keyword [S12a]

- Compiler Attributes can apply to functions
 - Inline – Skips calling convention, copies function body into calling code

```
__attribute__((always_inline)) inline int32_t add( int32_t x, int32_t y ) {  
    return (x + y);  
}
```


Inline keyword is a c99 Feature \Rightarrow Not supported in c89
always_inline is a GCC attribute \Rightarrow Not supported in other compilers

Inline Keyword [S12b]

- Compiler Attributes can apply to functions
 - Inline – Skips calling convention, copies function body into calling code

Compiler will
NOT ignore this

Compiler might
ignore this



```
__attribute__((always_inline)) inline int32_t add( int32_t x, int32_t y ) {  
    return (x + y);  
}
```

inline keyword is a c99 Feature \Rightarrow Not supported in c89


always_inline is a GCC attribute \Rightarrow Not supported on other compilers

Inline Keyword [S12c]

- Compiler Attributes can apply to functions
 - Inline – Skips calling convention, copies function body into calling code

Compiler will
NOT ignore this

Compiler might
ignore this



```
__attribute__((always_inline)) inline int32_t add( int32_t x, int32_t y ) {  
    return (x + y);  
}
```

Good for

- Small functions

Bad for / Will not work

for

- Recursive functions
- Variadic functions

Static Keyword [S13a]

- Static Keyword can apply to functions to create private access

Static Keyword [S13b]

- Static Keyword can apply to functions to create private access
 - Function/Variable only visible to current file
core_cm4.h

```
__attribute__( ( always_inline ) ) static inline void  
__enable_irq(void)  
{  
    __ASM volatile ("cpsie i" : : : "memory");  
}
```

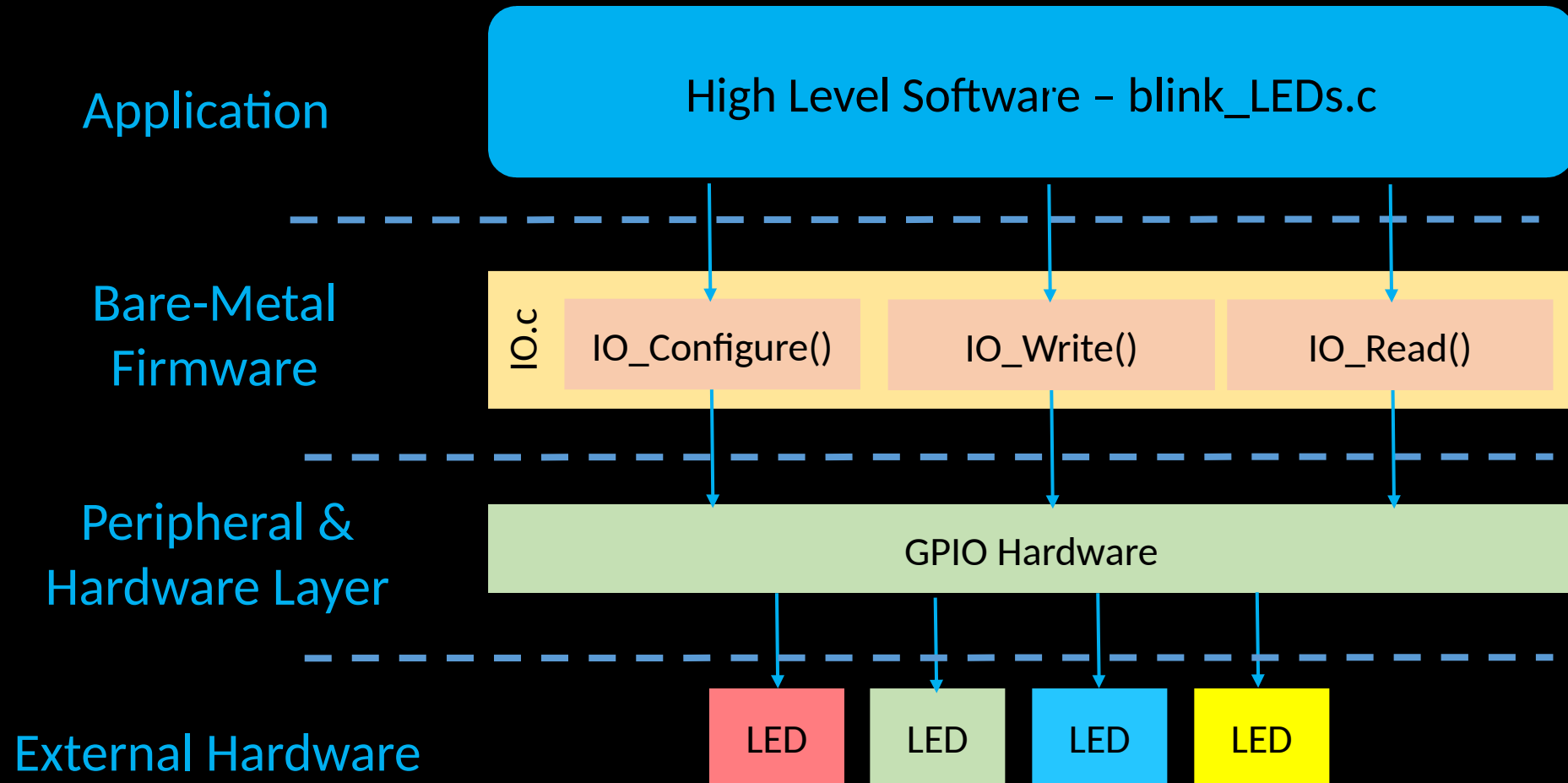
Static Keyword [S13c]

- Static Keyword can apply to functions to create private access
 - Function/Variable only visible to current file
core_cm4.h

```
__attribute__( ( always_inline ) ) static inline void  
__enable_irq(void)  
{  
    __ASM volatile ("cpsie i" : : : "memory");  
}
```

- Combine Static and Inline to prevent integrating all code

Interface Library for GPIO [S14a]

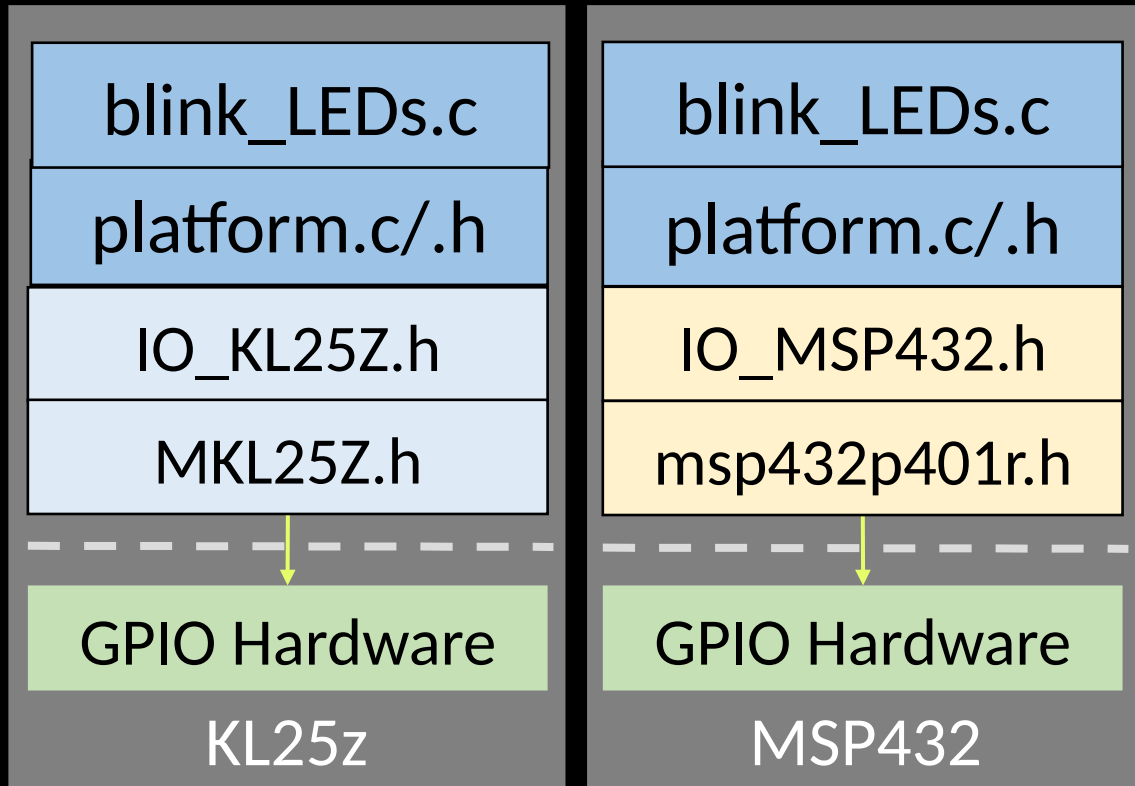


Interface Library for GPIO [S14b]

IO_MSP432.h

```
#include <stdio.h>
#include "msp.h"
__attribute__((always_inline)) static inline void
IO_Read(DIO_PORT_Type * port,
                                                uint8_t
pin)
{
    return ( ((port)->IN) & (1 << pin) );
}
__attribute__((always_inline)) static inline void IO_Write(DIO_PORT_Type * port,
                                                            uint8_t pin,
                                                            uint8_t value)
{
    value ? ( ((port)->OUT) |= (1 << pin) ) : ( ((port)->OUT) &= ~(1 << pin) );
}
```

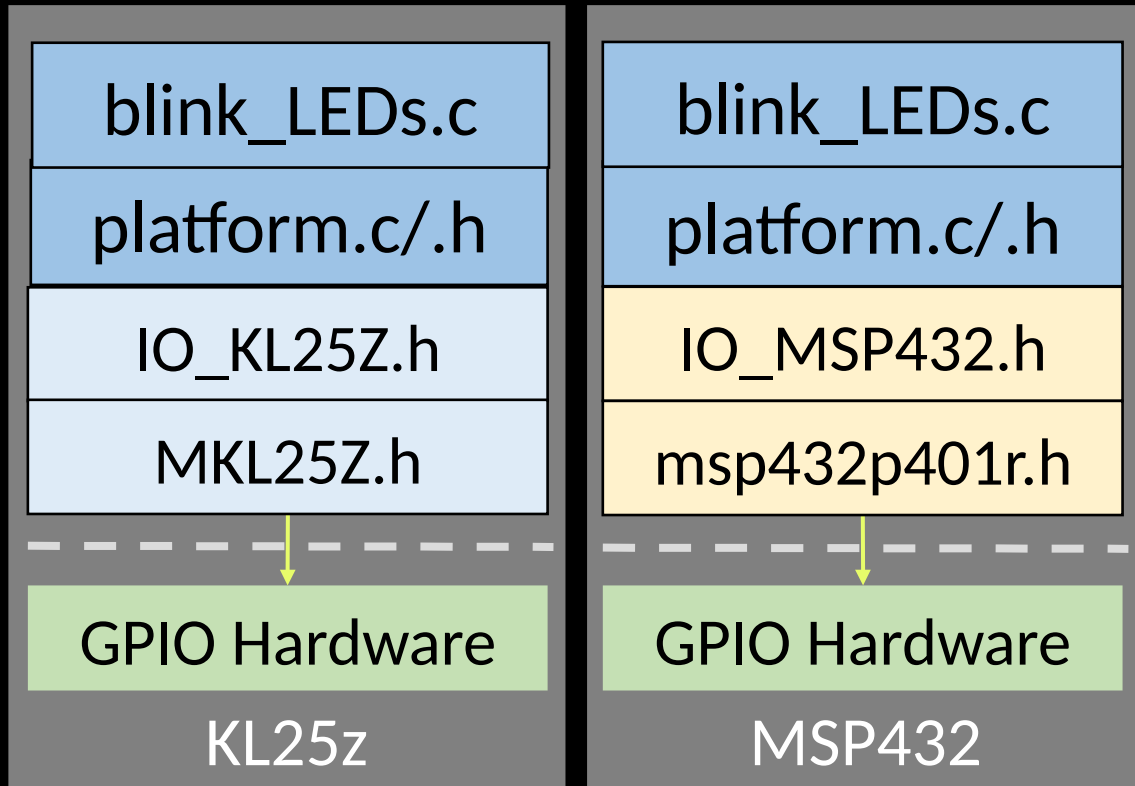
Interface Library for GPIO [S14c]



```
#include <stdio.h>
#include "msp.h"
__attribute__((always_inline)) static inline void
IO_Read(DIO_PORT_Type * port,
uint8_t pin)
{
    return ( ((port)->IN) & (1 << pin) );
}
__attribute__((always_inline)) static inline void IO_Write(DIO_PORT_Type * port,
                                                            uint8_t pin,
                                                            uint8_t value)
{
    value ? ( ((port)->OUT) |= (1 << pin) ) : ( ((port)->OUT) &= ~(1 << pin) );
}
```

IO_MSP432.h

Interface Library for GPIO [S14c]



```
#if defined (KL25Z ) && ! defined (MSP432)
#include "IO_KL25Z.h"
#elif defined (MSP432) && ! defined (KL25Z)
#include "IO_MSP432.h"
#else
#error "Platform not properly specified"
#endif
```

Platform.h

`$ make all PLATFORM=MSP432`