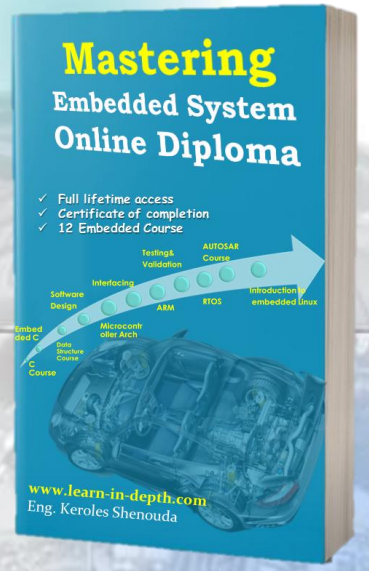
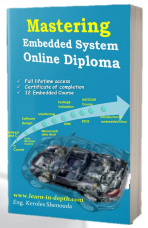
Mastering Embedded System Online Diploma

- ✓ Full lifetime access
- ✓ Access on Android mobile and PC (Windows)
- ✓ Certificate of completion
- ✓ 12 Embedded Course



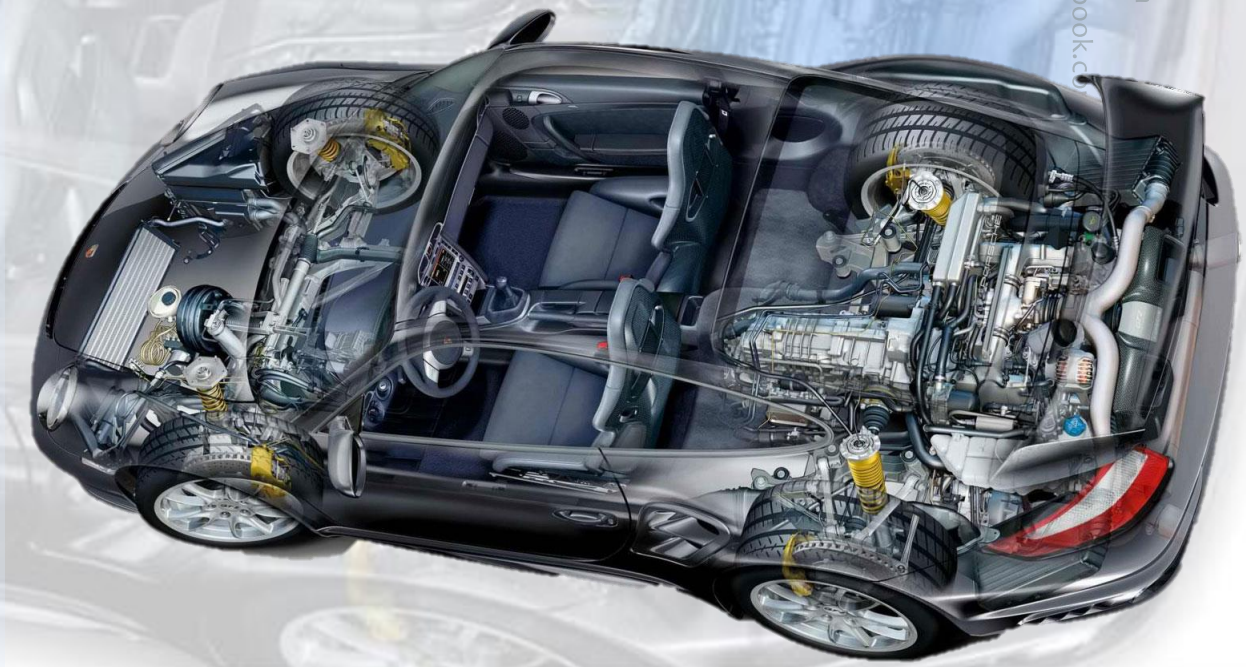


#LEARN_IN DEPTH

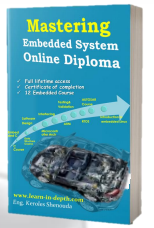
#Be_professional_in_embedded_system

Unit 6 (MCU Fundamentals) . lesson 2

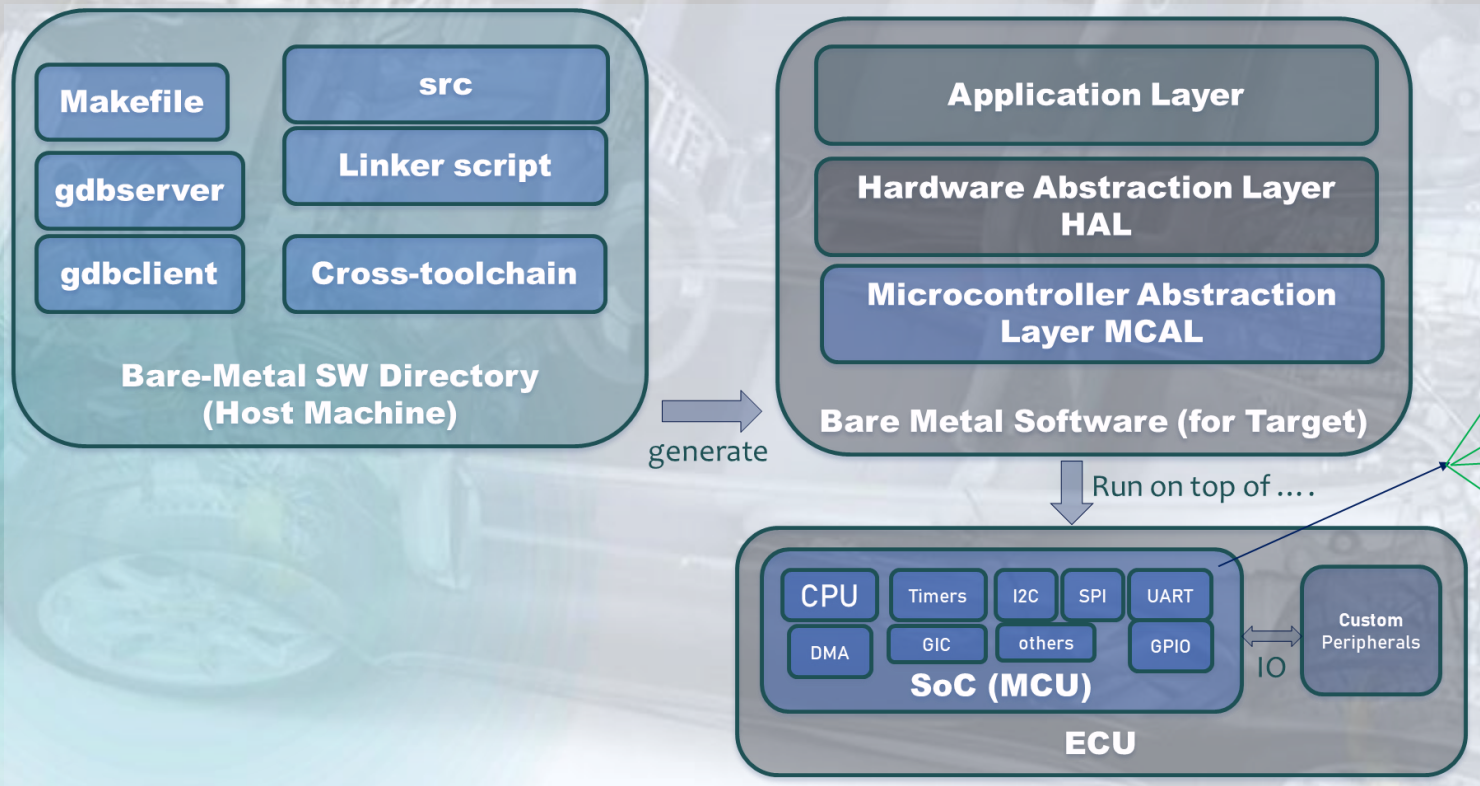
- port-mapped I/O, and memory-mapped I/O.
  - MCU Memory Map
    - Access Register Methods
      - *Method 1: Using numeric memory address directly*
      - *Method 2: Casting an address to a pointer*
      - *Method 3: Casting to a pointer and then dereferencing it*
      - Method 4: use structure, union and pointer for one register
      - The ARM Assembly == C "accessing registers" in ARM
    - Navigate "Memory map" for Stm32 in TRM
    - Navigate "Memory map" for TM4C123 in TRM
    - Figure out different based addresses from different MCUs
  - MCU Bus Interfaces
    - Understanding AMBA Bus Architecture and Protocols
    - AXI (Advanced Extensible Interface), Apb, ahb, … etc
    - Technical Questions about the BUS
    - BUS Matrix
- **Bit, Byte, Halfword, Word, Double-word and Nibbles**
- **Big and Little Endian**

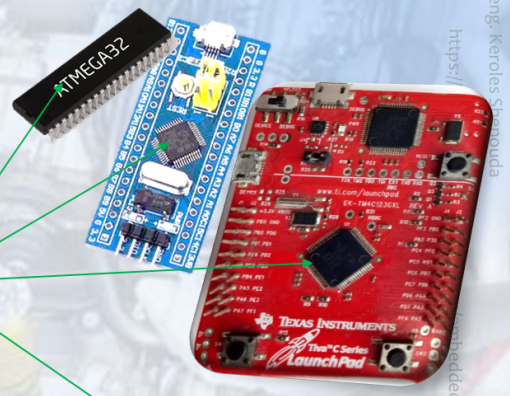


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Big Picture

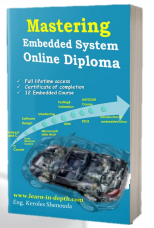




X – SoC

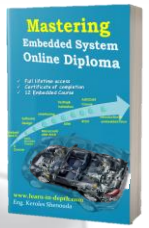
Read Specs and Program X - SoC

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



Unit 6 (Big Picture) Cont.

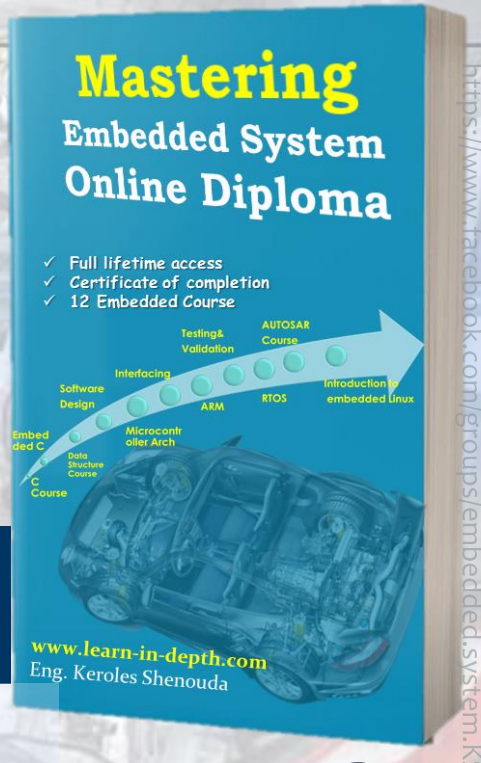
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

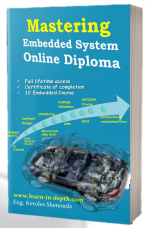
#Be_professional_in_embedded_system

port-mapped I/0, and memory-mapped I/0.



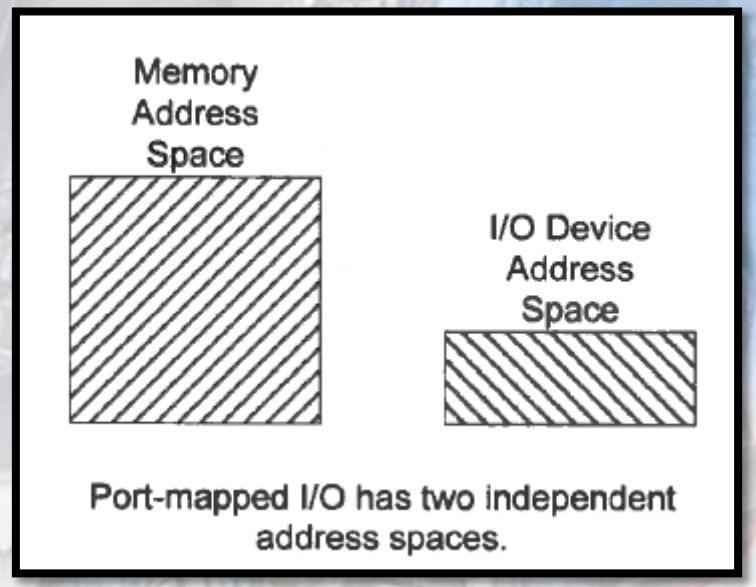
LEARN-IN-DEPTH
Be professional in embedded system

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



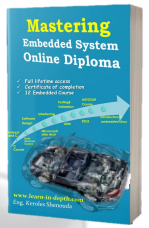
Port-mapped I/O

- uses special machine instructions, which are designed specifically for I/O operations.
- The memory address space and the I/O device address space are independent of each other.
- Each device is assigned one or more unique port numbers.
- For example, Intel x86 processors use IN and OUT instructions to read from or write to a port



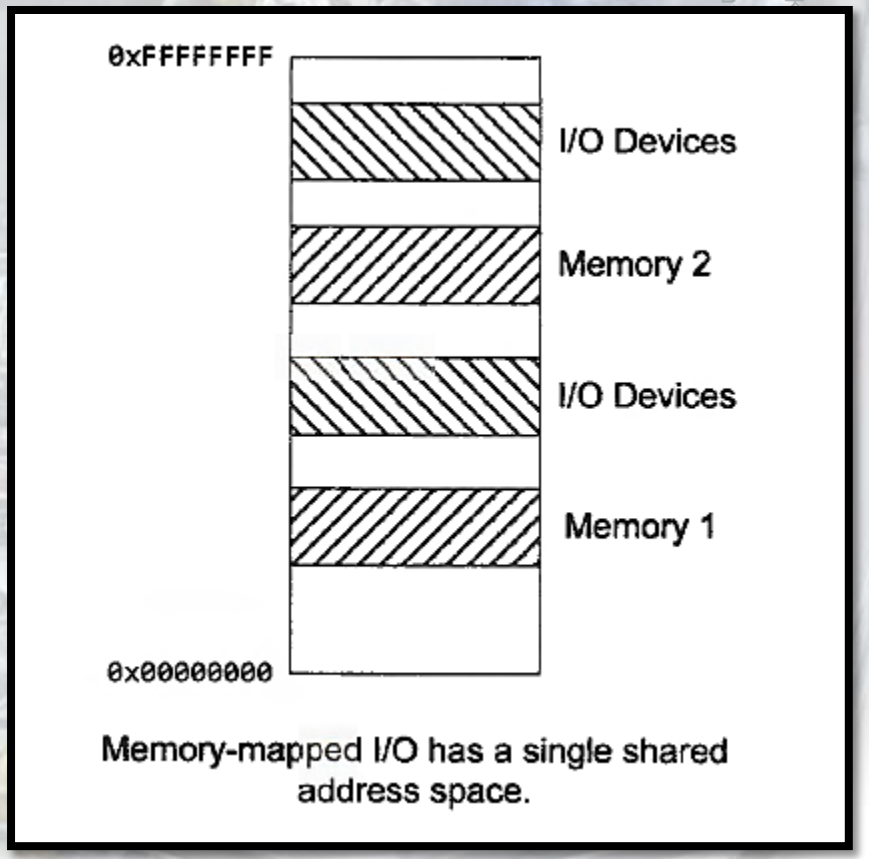
Port-mapped I/O has two independent address spaces.

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



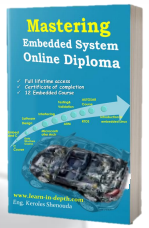
Memory-mapped I/O

- does not need any special instructions
- The memory and the I/O devices share the same address space.
- Each peripheral register or data buffer is assigned to a memory address in the memory address space of the microprocessor.
- Memory-mapped I/O is performed by the native load and store instructions of the processor.



Memory-mapped I/O has a single shared address space.

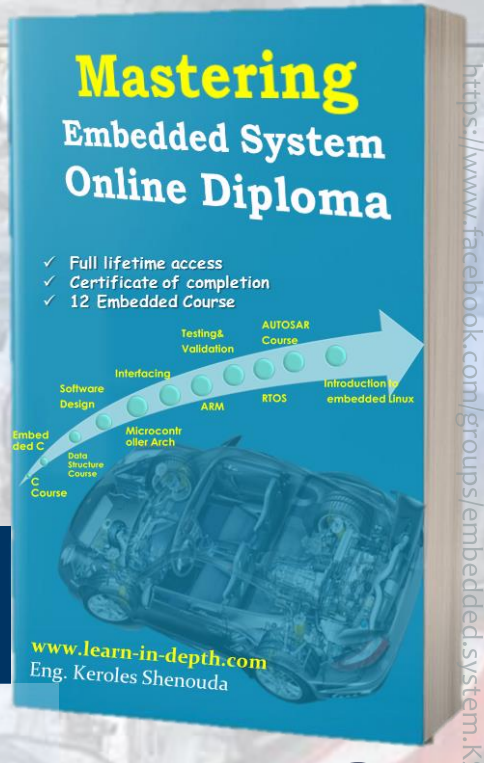
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

MCU Memory Map

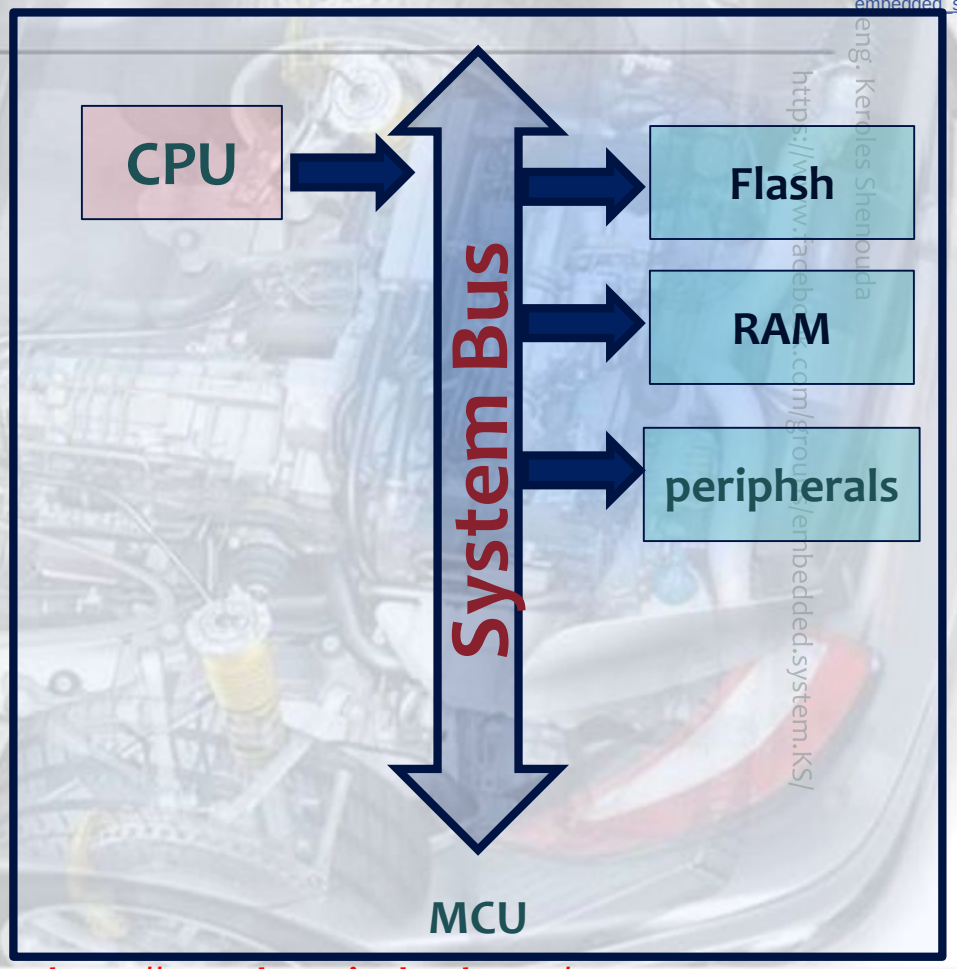


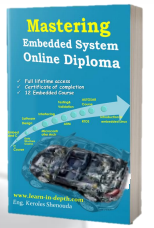
LEARN-IN-DEPTH
Be professional in embedded system

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

Memory map

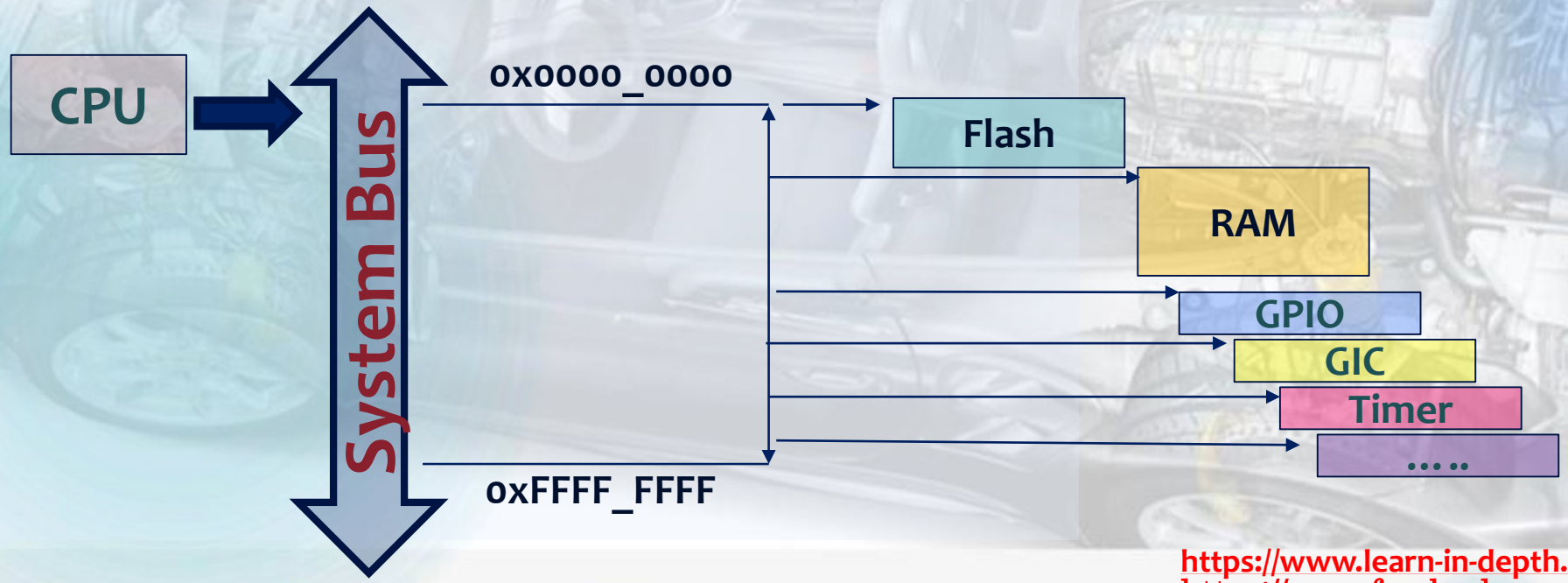
- If the system Bus is 32 bit
  - That means the CPU can issue transaction to 2^32 of different address (4G Bytes)





Memory map

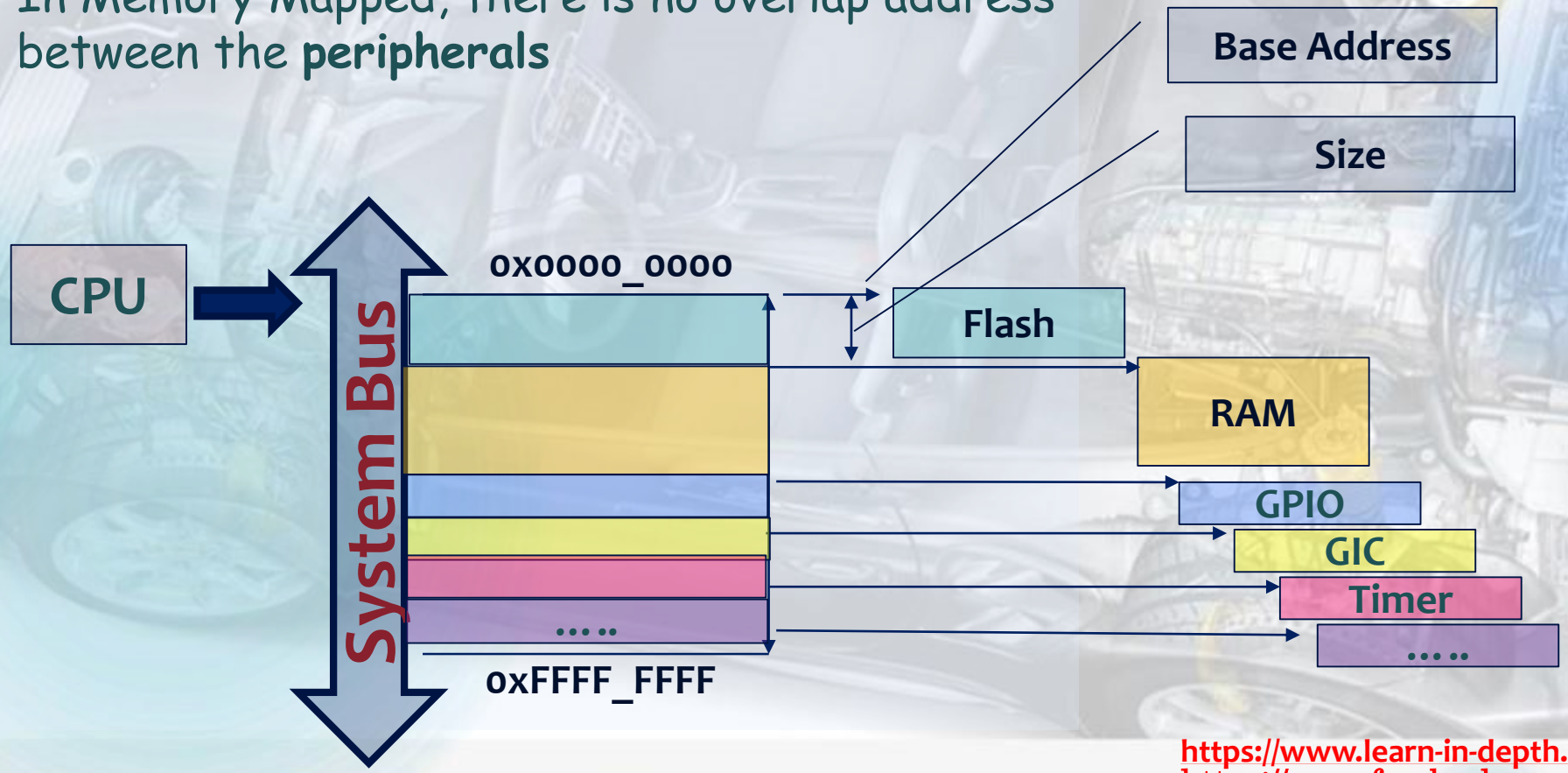
- If the system Bus is 32 bit
  - That means the CPU can issue transaction to 2^32 of different address (4G Bytes)



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

Memory map

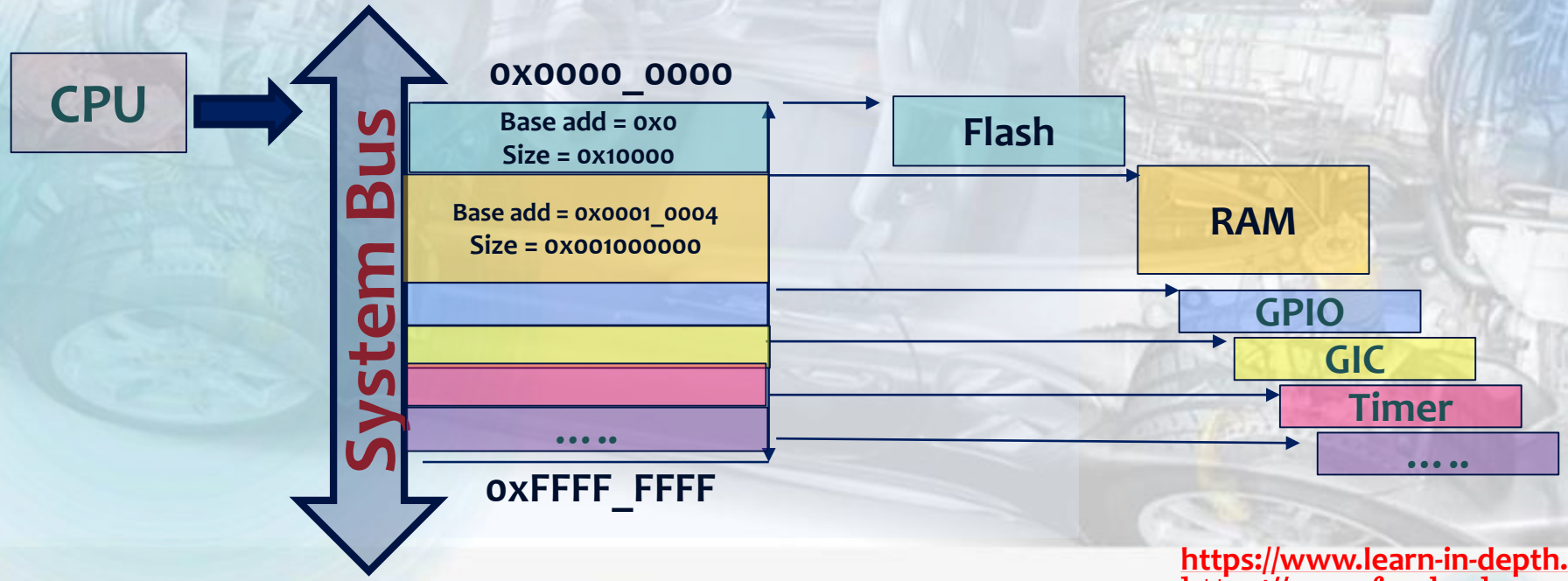
- In Memory Mapped, there is no overlap address between the **peripherals**



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

Memory map

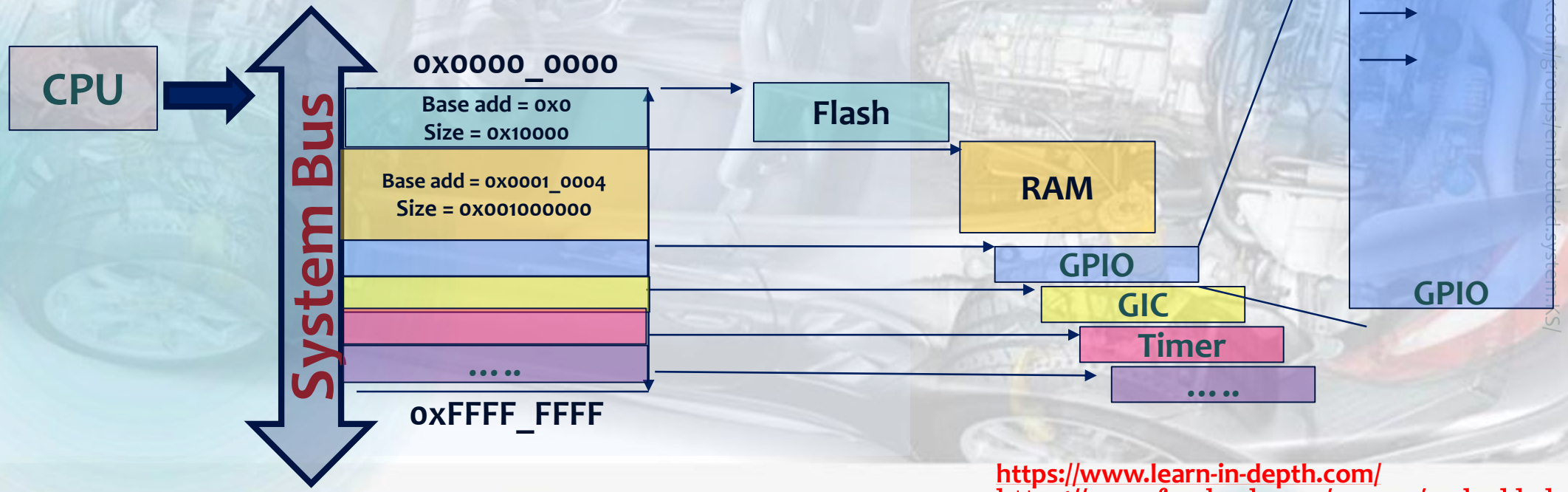
- If the system Bus is 32 bit
  - That means the CPU can issue transaction to 2^32 of different address (4G Bytes)



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

Memory map

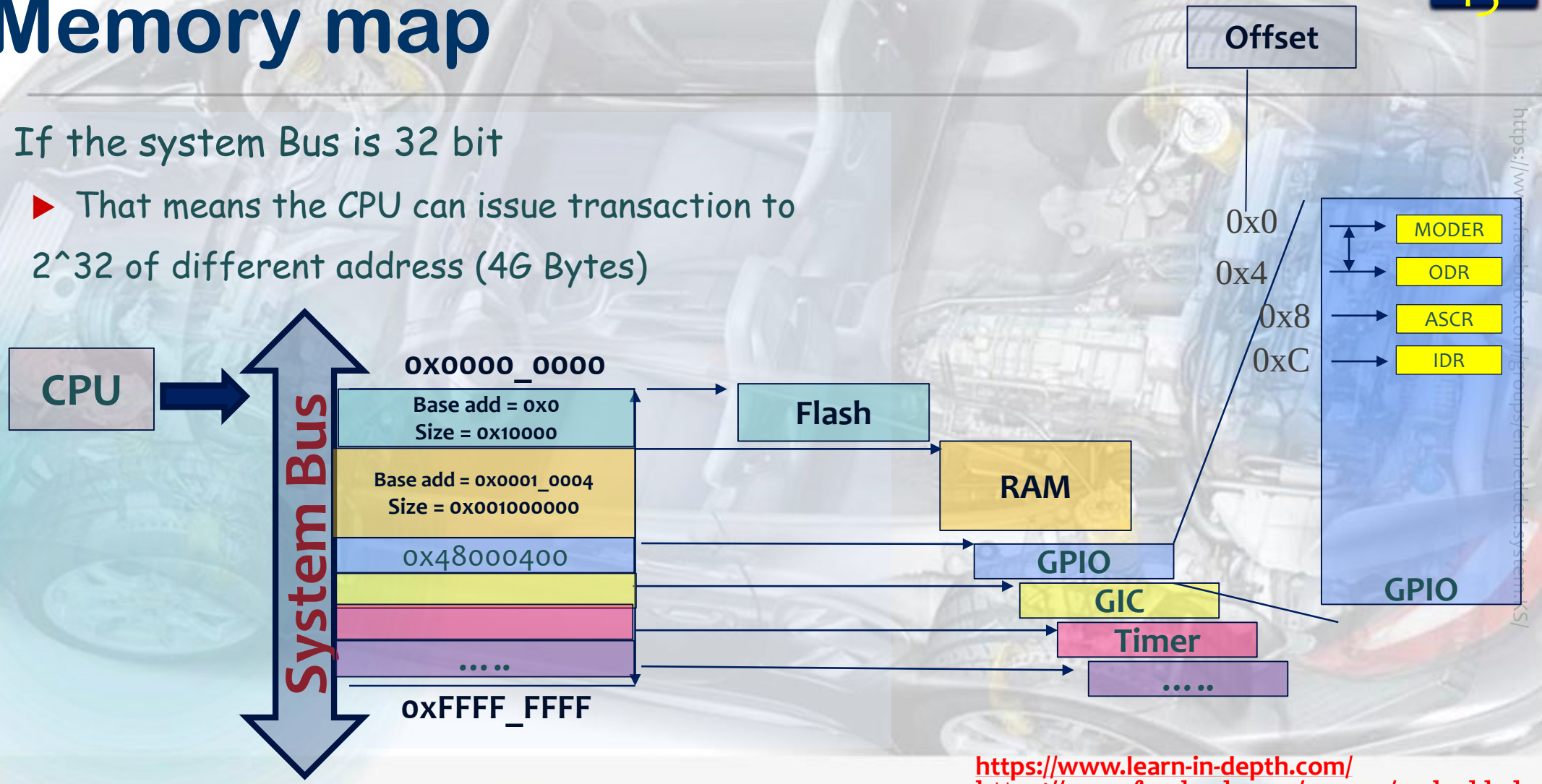
- If the system Bus is 32 bit
  - That means the CPU can issue transaction to 2^32 of different address (4G Bytes)

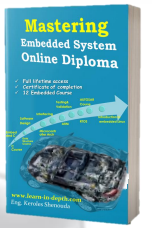


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

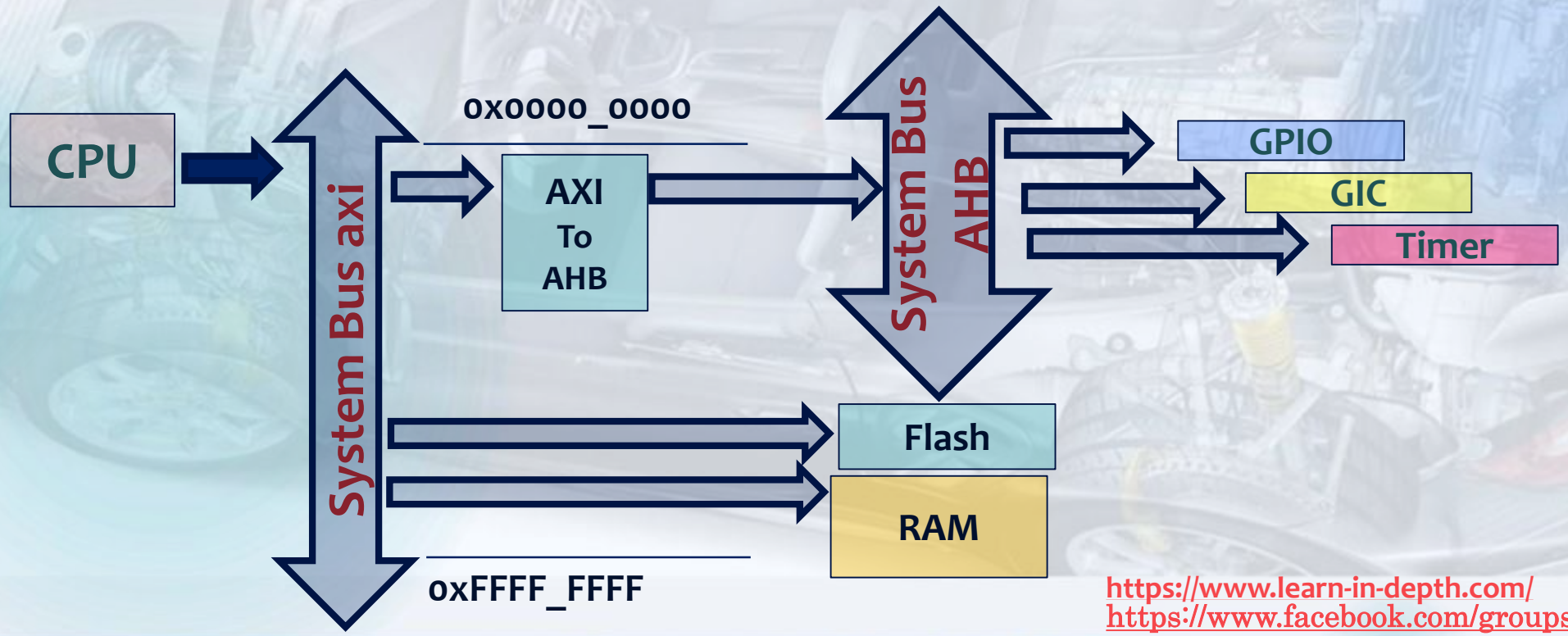
Memory map

- If the system Bus is 32 bit
  - That means the CPU can issue transaction to 2^32 of different address (4G Bytes)



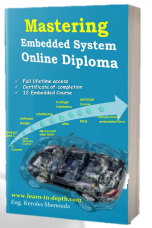


Bus Bridges



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

Question



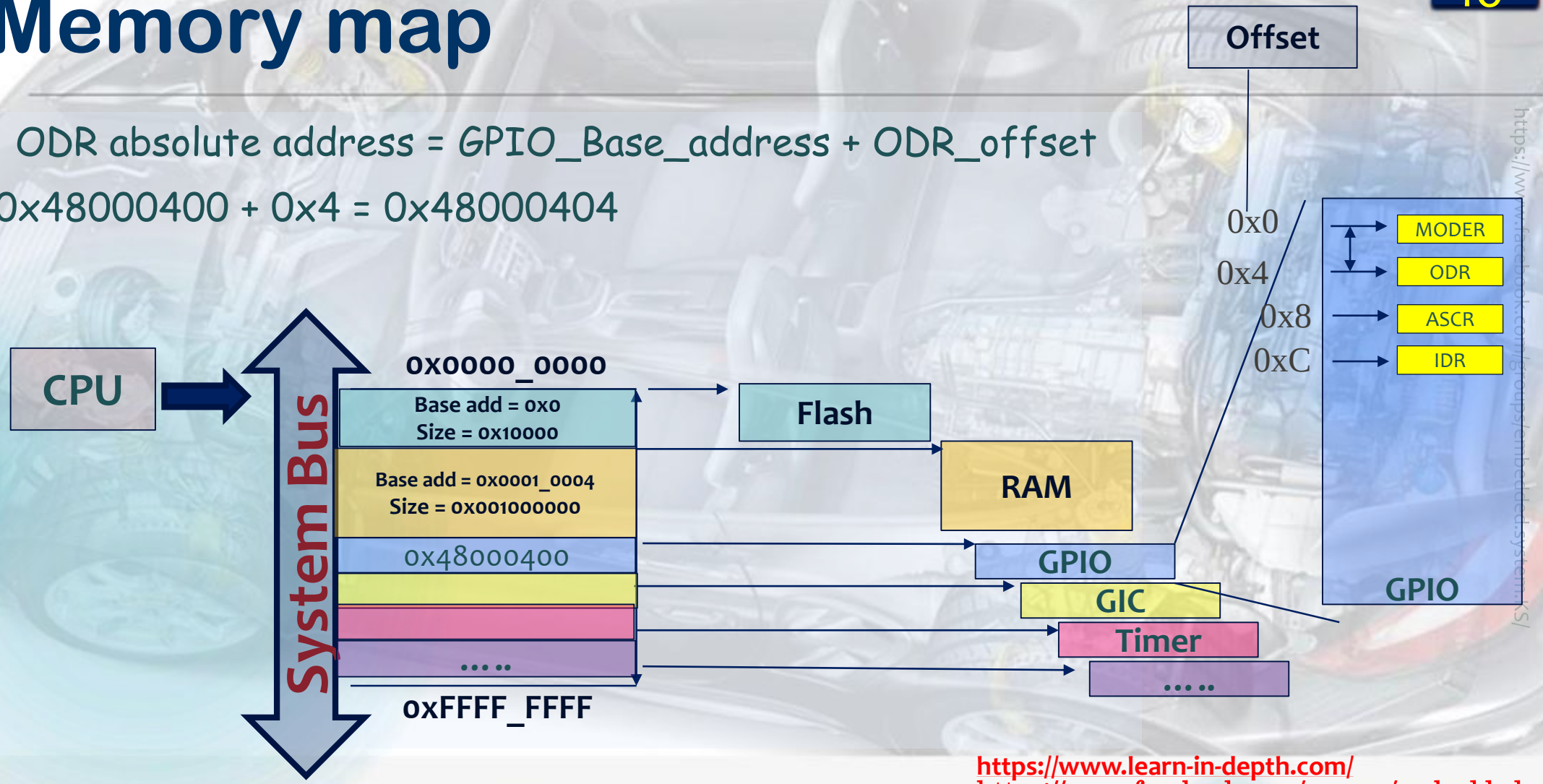
- Write 1 on pit 16 on ODR register by
  - *Method 1: Using numeric memory address directly*
  - *Method 2: Casting an address to a pointer*
  - *Method 3: Casting to a pointer and then dereferencing it*
  - Method 4: use structure, union and pointer for one register
- Method 5: use structures and pointers for all registers in a GPIO **peripheral**

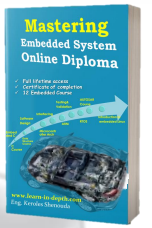
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Memory map

- ODR absolute address = GPIO_Base_address + ODR_offset

= 0x48000400 + 0x4 = 0x48000404



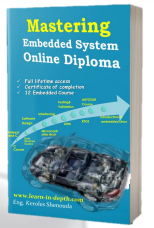


Method 1: Using numeric memory address directly

```c
/*
 *  Created on: Feb 24, 2021
 *      Author: Keroles Shenouda
 */

#include <stdio.h>
#include <stdint.h>

//just for testing in C on the host machine
uint32_t G_GPIO_ODR_variable = 0;
#define GPIO_ODR  &G_GPIO_ODR_variable

//on target just use the absolute address
//#define GPIO_ODR  0x48000404

void main ()
{

    //Method 1: Using numeric memory address directly
    *((volatile uint32_t *) GPIO_ODR) |= 1UL<<16 ;

    printf ("G_GPIO_ODR_variable = 0x%X",G_GPIO_ODR_variable);

}
```

```
<terminated> (exit value: 29) Unit6 Debug [C/C++ Application] D:\courses\new_diploma\Diploma online\Eclipse_ws\Unit6\Debug\Unit6.exe (2/24/21, 10:22 PM)
G_GPIO_ODR_variable = 0x10000
```

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



Method 2: Casting an address to a pointer

main.c

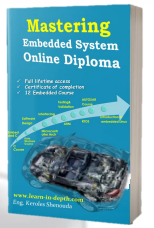
```
 *  Created on: Feb 24, 2021
 *      Author: Keroles Shenouda
 */

#include <stdio.h>
#include <stdint.h>

//just for testing in C on the host machine
uint32_t G_GPIO_ODR_variable = 0;
#define GPIO_ODR_Add  &G_GPIO_ODR_variable

#define GPIO_ODR ((volatile uint32_t *) GPIO_ODR_Add)
//on target just use the absolute address
//#define GPIO_ODR  0x48000404

void main ()
{

    //Method 1: Using numeric memory address directly
    *GPIO_ODR |= 1UL<<16 ;

    printf ("G_GPIO_ODR_variable = 0x%X",G_GPIO_ODR_variable);

}
```

Problems Tasks Console Properties Console Debug

<terminated> (exit value: 29) Unit6 Debug [C/C++ Application] D:\courses\new_diploma\Diploma online\Eclipse_ws\Unit6\Debug\Unit6.exe (2/24/21, 10:27 PM)

```
G_GPIO_ODR_variable = 0x10000
```


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



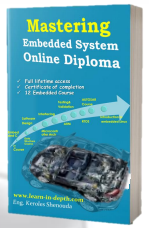
Method 3: Casting to a pointer and then dereferencing it

```c
main.c
 *  Created on: Feb 24, 2021
 *      Author: Keroles Shenouda
 */

#include <stdio.h>
#include <stdint.h>

//just for testing in C on the host machine
uint32_t G_GPIO_ODR_variable = 0;
#define GPIO_ODR_Add  &G_GPIO_ODR_variable

#define GPIO_ODR (*((volatile uint32_t *) GPIO_ODR_Add))
//on target just use the absolute address
//#define GPIO_ODR  0x48000404

void main ()
{

    //Method 3: Casting to a pointer and then dereferencing it
    GPIO_ODR |= 1UL<<16 ;

    printf ("G_GPIO_ODR_variable = 0x%X",G_GPIO_ODR_variable);

}
```

```
Problems  Tasks  Console  Properties  Console  Debug
<terminated> (exit value: 29) Unit6 Debug [C/C++ Application] D:\courses\new_diploma\Diploma online\Eclipse_ws\Unit6\Debug\Unit6.exe (2/24/21, 10:29 PM)
G_GPIO_ODR_variable = 0x10000
```


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

Method 4: use structure, union and pointer for one register



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

main.c

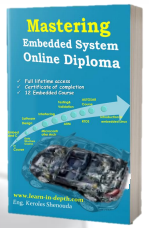
```c
/*
 * main.c
 *
 *  Created on: Feb 24, 2021
 *      Author: Keroles Shenouda
 */

#include <stdio.h>
#include <stdint.h>

//just for testing in C on the host machine
uint32_t G_GPIO_ODR_variable = 0;
#define GPIO_ODR_Add  &G_GPIO_ODR_variable

#define GPIO_ODR (*((volatile uint32_t *) GPIO_ODR_Add))
//on target just use the absolute address
//#define GPIO_ODR  0x48000404

#pragma pack(1)
struct SGPIO_ODR_t
{
    uint32_t pin0 :1 ;
    uint32_t pin1 :1 ;
    uint32_t pin2 :1 ;
    uint32_t pin3 :1 ;
    uint32_t pin4 :1 ;
    uint32_t pin5 :1 ;
    uint32_t pin6 :1 ;
    uint32_t pin7 :1 ;
    uint32_t pin8_ :8 ;
    uint32_t pin16 :1 ;
    uint32_t pin17 :1 ;
    uint32_t pin18 :1 ;
    uint32_t pin19 :1 ;
    uint32_t pin20 :12 ;
};
```

```c
union U_odr {
    struct SGPIO_ODR_t S_ODR ;
    uint32_t ODR ;
};

volatile union U_odr* GPIO_ODR_R = (union U_odr*) GPIO_ODR_Add ;

void main ()
{

//  Method 4: use structure, union  and pointer for one register
    GPIO_ODR_R->S_ODR.pin16 = 1 ;

    printf ("G_GPIO_ODR_variable = 0x%X",G_GPIO_ODR_variable);

}
```

Problems Tasks Console Properties Console Debug

```
<terminated> (exit value: 29) Unit6 Debug [C/C++ Application] D:\courses\new_diploma\Diploma online\Eclipse_ws\Unit6\Debug\Unit6.exe (2/24/21, 10:42 P
G_GPIO_ODR_variable = 0x10000
```

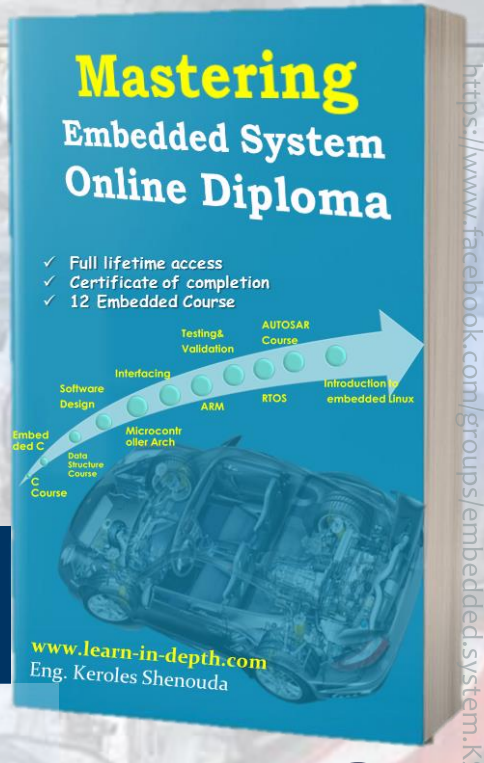
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

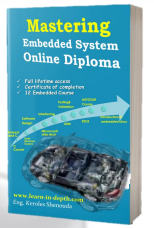
#Be_professional_in_embedded_system

The ARM Assembly == C “accessing registers” in ARM

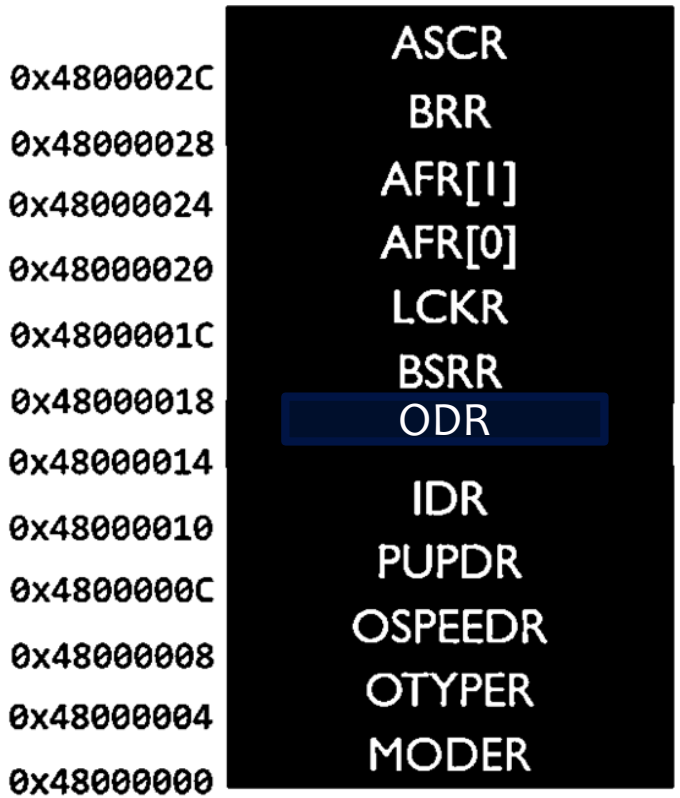


LEARN-IN-DEPTH
Be professional in embedded system

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

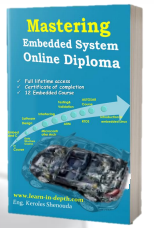


Another example




```
typedef struct {
  volatile uint32_t MODER;    // Mode register
  volatile uint32_t OTYPER;   // Output type register
  volatile uint32_t OSPEEDR;  // Output speed register
  volatile uint32_t PUPDR;    // Pull-up/pull-down register
  volatile uint32_t IDR;      // Input data register
  volatile uint32_t ODR;      // Output data register
  volatile uint32_t BSRR;     // Bit set/reset register
  volatile uint32_t LCKR;     // Configuration lock register
  volatile uint32_t AFR[2];   // Alternate function registers
  volatile uint32_t BRR;      // Bit Reset register
  volatile uint32_t ASCR;     // Analog switch control register
} GPIO_TypeDef;

// Casting memory address to a pointer
#define GPIOA ((GPIO_TypeDef *) 0x48000000)
```


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



Another example

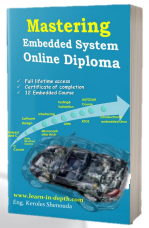
In C

```
GPIOA->ODR |= 1UL<<14;    // Set bit 14 to 1
```

In Assembly

```
GPIOA_BASE EQU   0x48000000
GPIO_ODR   EQU   0x14

LDR r7, =GPIOA_BASE       ; Load GPIO port A base address
LDR r1, [r7, #GPIO_ODR]   ; r1 = GPIOA->ODR
ORR r1, r1, #(1<<14)      ; Set output of pin 14 to high
STR r1, [r7, #GPIO_ODR]   ; Write the output data register
```

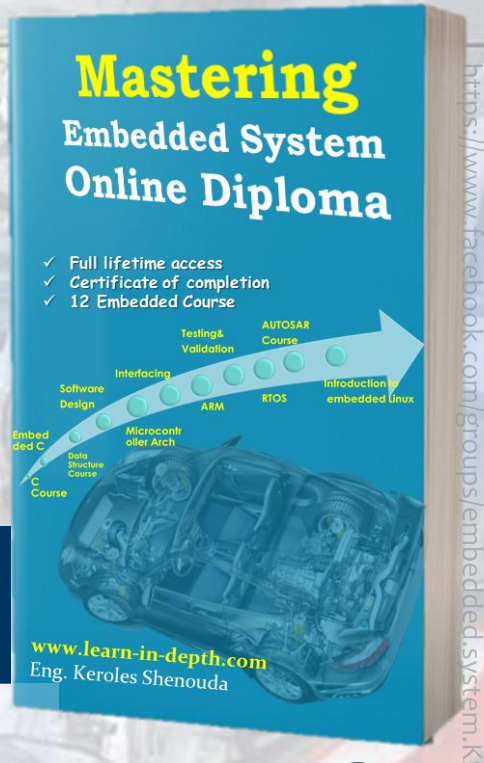

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

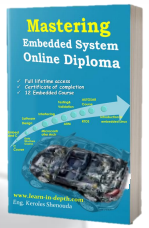
#Be_professional_in_embedded_system

Method 5: use structures and pointers to access all registers in a Any peripheral



LEARN-IN-DEPTH
Be professional in embedded system

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



Question

- Write 1 on pit 16 on ODR register by
  - *Method 1: Using numeric memory address directly*
  - *Method 2: Casting an address to a pointer*
  - *Method 3: Casting to a pointer and then dereferencing it*
  - Method 4: use structure, union and pointer for one register
- **Method 5: use structures and pointers for all registers in a GPIO peripheral**

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

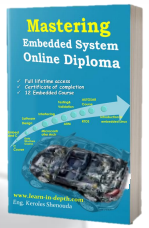
Method 5: use structures and pointers for all registers in a GPIO peripheral

```
main.c
/*
 * main.c
 *
 *  Created on: Feb 24, 2021
 *      Author: Keroles Shenouda
 */

#include <stdio.h>
#include <stdint.h>

//just for testing in C on the host machine
uint32_t G_GPIO_variable[4] = {0};
#define GPIO_B_Add  &G_GPIO_variable[0]


#define __IO volatile // allows read and write

#pragma pack(1)
struct SGPIO {
    __IO uint32_t MODER ; //Mode register
    __IO uint32_t ODR ; //Output data register
    __IO uint32_t ASCR ; //Analog switch control register
    __IO uint32_t IDR ; //Input data register

};

#define GPIOB ((struct SGPIO*) GPIO_B_Add)

//#define GPIOA ((GPIO_TypeDef *) 0x48000000))
//#define GPIOB ((GPIO_TypeDef *) 0x48000400))
//#define GPIOC ((GPIO_TypeDef *) 0x40000800))

void main ()
{
    //  Method 5: use structures and pointers for all registers in a GPIO peripheral
    GPIOB->ODR = (1UL << 16) ;
    //GPIO_Init(GPIOA);
    //GPIO_Init(GPIOB);


    printf ("G_GPIO_ODR_variable = 0x%X",G_GPIO_variable[1]);

}
```

Problems | Tasks | Console | Properties

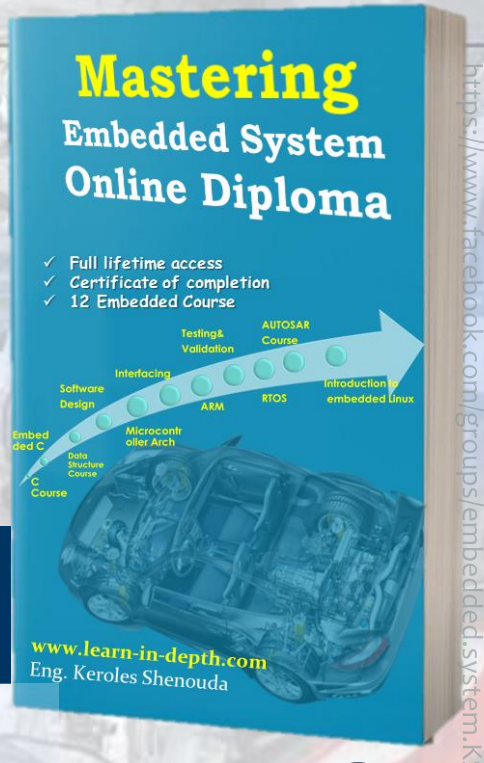
<terminated> (exit value: 29) Unit6 Debug [C/C++ Application] D

```
G_GPIO_ODR_variable = 0x10000
```



#LEARN_IN DEPTH

#Be_professional_in_embedded_system



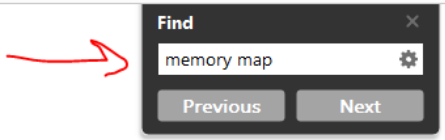
Navigate “Memory map” for Stm32 in TRM

TRM: TECHNICAL REFERENCE MANUAL

LEARN-IN-DEPTH
Be professional in embedded system

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

Navigate “Memory map” in Stm32



life.augmented

RM0008
Reference manual

STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced Arm®-based 32-bit MCUs

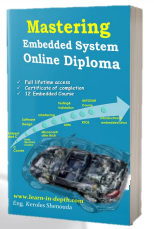
Introduction

This reference manual is addressed to application developers.

It provides complete information on how to use the STM32F101xx, STM32F102xx, STM32F103xx and STM32F105xx/STM32F107xx microcontroller memory and peripherals. The STM32F101xx, STM32F102xx, STM32F103xx and STM32F105xx/STM32F107xx will be referred to as STM32F10xxx throughout the document, unless otherwise specified.

The STM32F10xxx is a family of microcontrollers with different memory sizes, packages

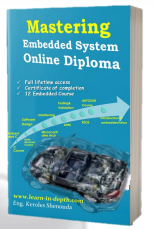
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Navigate “Memory map” in Stm32

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

Navigate “Memory map” in Stm32



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

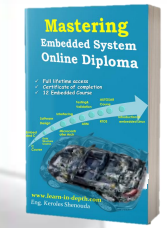
3.3 Memory map

See the datasheet corresponding to your device for a comprehensive diagram of the memory map. *Table 3* gives the boundary addresses of the peripherals available in all STM32F10xxx devices.

Table 3. Register boundary addresses

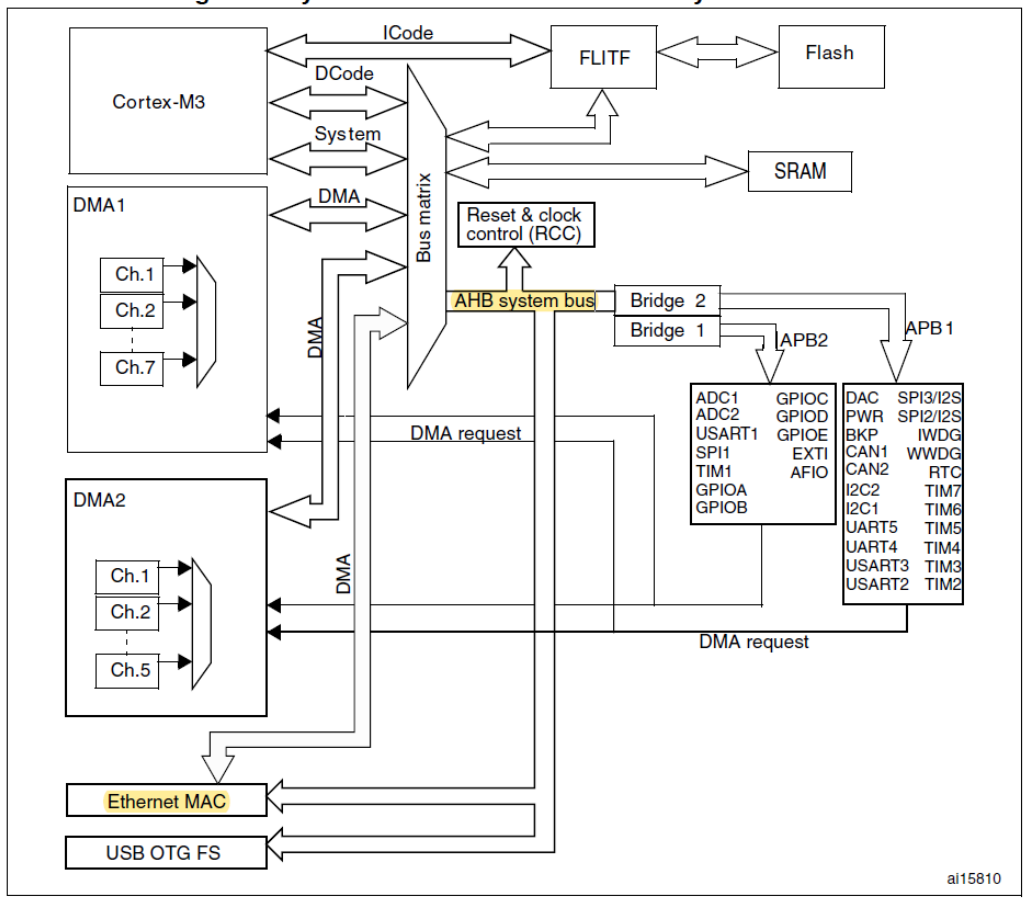
Boundary address	Peripheral	Bus	Register map
0xA000 0000 - 0xA000 0FFF	FSMC	AHB	*Section 21.6.9 on page 564*
0x5000 0000 - 0x5003 FFFF	USB OTG FS		*Section 28.16.6 on page 913*
0x4003 0000 - 0x4FFF FFFF	Reserved		-
0x4002 8000 - 0x4002 9FFF	Ethernet		*Section 29.8.5 on page 1069*
0x4002 3400 - 0x4002 7FFF	Reserved		-
0x4002 3000 - 0x4002 33FF	CRC		*Section 4.4.4 on page 65*
0x4002 2000 - 0x4002 23FF	Flash memory interface		-
0x4002 1400 - 0x4002 1FFF	Reserved		-
0x4002 1000 - 0x4002 13FF	Reset and clock control RCC		*Section 7.3.11 on page 121*
0x4002 0800 - 0x4002 0FFF	Reserved		-
0x4002 0400 - 0x4002 07FF	DMA2		*Section 13.4.7 on page 289*
0x4002 0000 - 0x4002 03FF	DMA1		
0x4001 8400 - 0x4001 FFFF	Reserved		-
0x4001 8000 - 0x4001 83FF	SDIO		*Section 22.9.16 on page 621*

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



Navigate “Memory map” in Stm32

Figure 2. System architecture in connectivity line devices



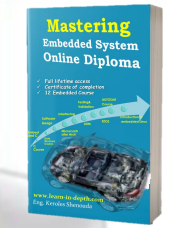
3.3 Memory map

See the datasheet corresponding to your device for a comprehensive diagram of the memory map. *Table 3* gives the boundary addresses of the peripherals available in all STM32F10xxx devices.

Table 3. Register boundary addresses

Boundary address	Peripheral	Bus	Register map
0xA000 0000 - 0xA000 0FFF	FSMC	AHB	*Section 21.6.9 on page 564*
0x5000 0000 - 0x5003 FFFF	USB OTG FS		*Section 28.16.6 on page 913*
0x4003 0000 - 0x4FFF FFFF	Reserved		-
0x4002 8000 - 0x4002 9FFF	Ethernet		*Section 29.8.5 on page 1069*
0x4002 3400 - 0x4002 7FFF	Reserved		-
0x4002 3000 - 0x4002 33FF	CRC		*Section 4.4.4 on page 65*
0x4002 2000 - 0x4002 23FF	Flash memory interface		-
0x4002 1400 - 0x4002 1FFF	Reserved		-
0x4002 1000 - 0x4002 13FF	Reset and clock control RCC		*Section 7.3.11 on page 121*
0x4002 0800 - 0x4002 0FFF	Reserved		-
0x4002 0400 - 0x4002 07FF	DMA2		*Section 13.4.7 on page 289*
0x4002 0000 - 0x4002 03FF	DMA1		
0x4001 8400 - 0x4001 FFFF	Reserved		-
0x4001 8000 - 0x4001 83FF	SDIO		*Section 22.9.16 on page 621*

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



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

Navigate "Memory map" in Stm32

Figure 2. System architecture in connectivity line devices

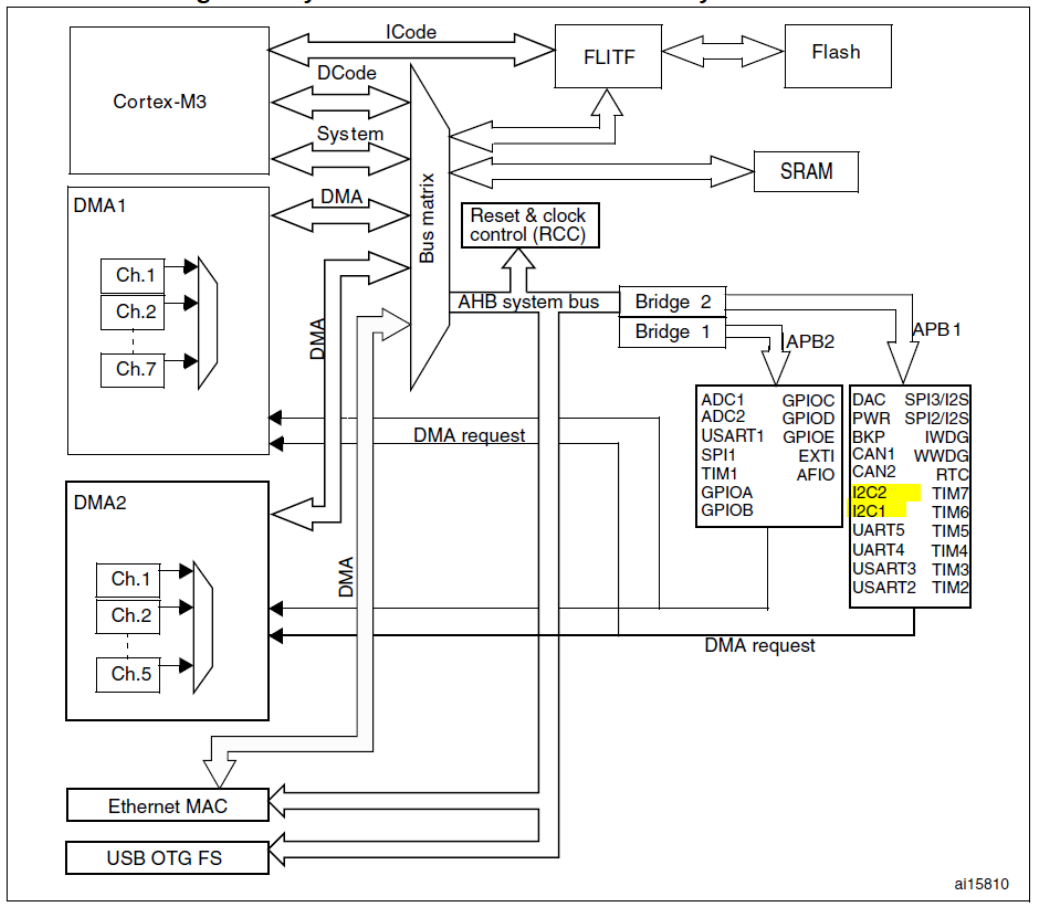
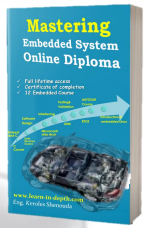


Table 3. Register boundary addresses (continued)

Boundary address	Peripheral	Bus	Register map
0x4000 7800 - 0x4000 FFFF	Reserved		-
0x4000 7400 - 0x4000 77FF	DAC		*Section 12.5.14 on page 273*
0x4000 7000 - 0x4000 73FF	Power control PWR		*Section 5.4.3 on page 80*
0x4000 6C00 - 0x4000 6FFF	Backup registers (BKP)		*Section 6.4.5 on page 85*
0x4000 6400 - 0x4000 67FF	bxCAN1		*Section 24.9.5 on page 695*
0x4000 6800 - 0x4000 6BFF	bxCAN2		
0x4000 6000[(1)] - 0x4000 63FF	Shared USB/CAN SRAM 512 bytes		-
0x4000 5C00 - 0x4000 5FFF	USB device FS registers		*Section 23.5.4 on page 651*
0x4000 5800 - 0x4000 5BFF	I2C2		*Section 26.6.10 on page 784*
0x4000 5400 - 0x4000 57FF	I2C1		
0x4000 5000 - 0x4000 53FF	UART5		*Section 27.6.8 on page 827*
0x4000 4C00 - 0x4000 4FFF	UART4		
0x4000 4800 - 0x4000 4BFF	USART3		
0x4000 4400 - 0x4000 47FF	USART2		
0x4000 4000 - 0x4000 43FF	Reserved		-
0x4000 3C00 - 0x4000 3FFF	SPI3/I2S	APB1	*Section 25.5 on page 742*
0x4000 3800 - 0x4000 3BFF	SPI2/I2S		*Section 25.5 on page 742*
0x4000 3400 - 0x4000 37FF	Reserved		-
0x4000 3000 - 0x4000 33FF	Independent watchdog (IWDG)		*Section 19.4.5 on page 499*
0x4000 2C00 - 0x4000 2FFF	Window watchdog (WWDG)		*Section 20.6.4 on page 506*
0x4000 2800 - 0x4000 2BFF	RTC		*Section 18.4.7 on page 493*
0x4000 2400 - 0x4000 27FF	Reserved		-
0x4000 2000 - 0x4000 23FF	TIM14 timer		*Section 16.5.11 on page 468*
0x4000 1C00 - 0x4000 1FFF	TIM13 timer		
0x4000 1800 - 0x4000 1BFF	TIM12 timer		*Section 16.4.13 on page 458*
0x4000 1400 - 0x4000 17FF	TIM7 timer		*Section 17.4.9 on page 481*

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



Navigate “Memory map” in Stm32

Figure 2. System architecture in connectivity line devices

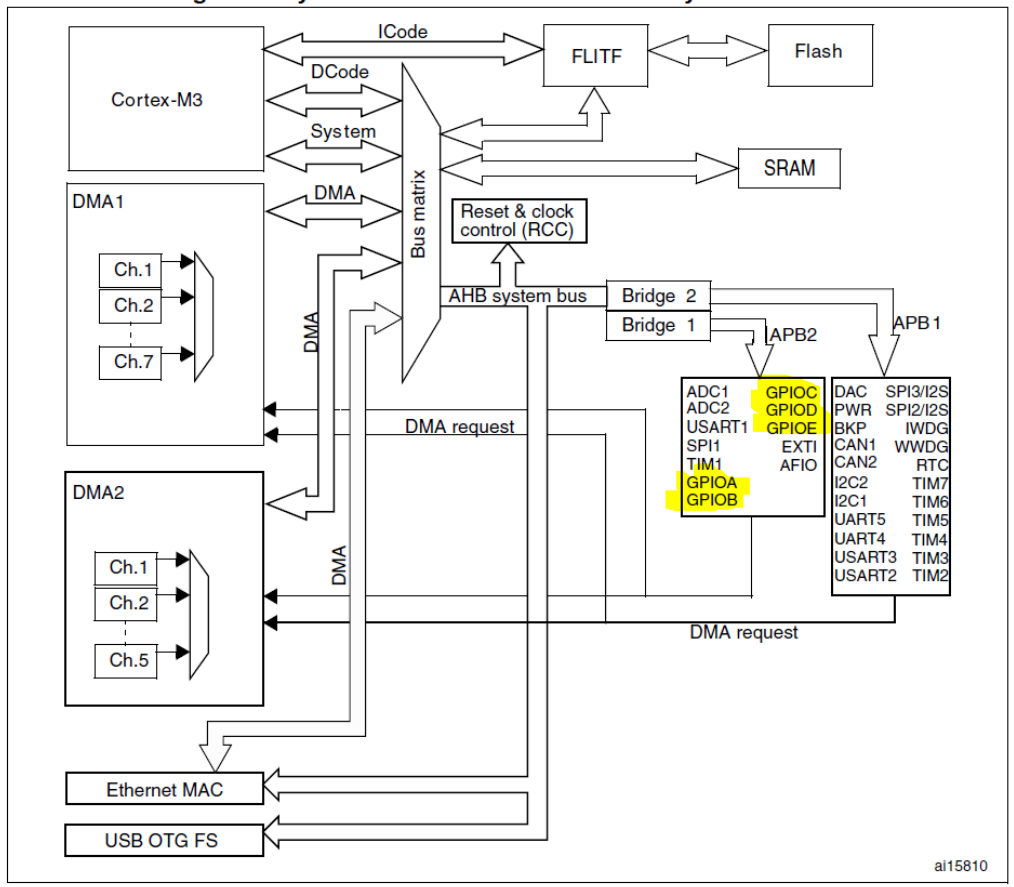


Table 3. Register boundary addresses (continued)

Boundary address	Peripheral	Bus	Register map
0x4001 5800 - 0x4001 7FFF	Reserved	APB2	-
0x4001 5400 - 0x4001 57FF	TIM11 timer		Section 16.5.11 on page 468
0x4001 5000 - 0x4001 53FF	TIM10 timer		Section 16.5.11 on page 468
0x4001 4C00 - 0x4001 4FFF	TIM9 timer		Section 16.4.13 on page 458
0x4001 4000 - 0x4001 4BFF	Reserved		-
0x4001 3C00 - 0x4001 3FFF	ADC3		Section 11.12.15 on page 252
0x4001 3800 - 0x4001 3BFF	USART1		Section 27.6.8 on page 827
0x4001 3400 - 0x4001 37FF	TIM8 timer		Section 14.4.21 on page 363
0x4001 3000 - 0x4001 33FF	SPI1		Section 25.5 on page 742
0x4001 2C00 - 0x4001 2FFF	TIM1 timer		Section 14.4.21 on page 363
0x4001 2800 - 0x4001 2BFF	ADC2		Section 11.12.15 on page 252
0x4001 2400 - 0x4001 27FF	ADC1		
0x4001 2000 - 0x4001 23FF	GPIO Port G		Section 9.5 on page 194
0x4001 1C00 - 0x4001 1FFF	GPIO Port F		
0x4001 1800 - 0x4001 1BFF	GPIO Port E		
0x4001 1400 - 0x4001 17FF	GPIO Port D		
0x4001 1000 - 0x4001 13FF	GPIO Port C		
0x4001 0C00 - 0x4001 0FFF	GPIO Port B		
0x4001 0800 - 0x4001 0BFF	GPIO Port A		
0x4001 0400 - 0x4001 07FF	EXTI		Section 10.3.7 on page 214
0x4001 0000 - 0x4001 03FF	AFIO		Section 9.5 on page 194

Navigate “Memory map” in Stm32

Table 3. Register boundary addresses (continued)

Boundary address	Peripheral	Bus	Register map
0x4001 5800 - 0x4001 7FFF	Reserved	APB2	-
0x4001 5400 - 0x4001 57FF	TIM11 timer		Section 16.5.11 on page 468
0x4001 5000 - 0x4001 53FF	TIM10 timer		Section 16.5.11 on page 468
0x4001 4C00 - 0x4001 4FFF	TIM9 timer		Section 16.4.13 on page 458
0x4001 4000 - 0x4001 4BFF	Reserved		-
0x4001 3C00 - 0x4001 3FFF	ADC3		Section 11.12.15 on page 252
0x4001 3800 - 0x4001 3BFF	USART1		Section 27.6.8 on page 827
0x4001 3400 - 0x4001 37FF	TIM8 timer		Section 14.4.21 on page 363
0x4001 3000 - 0x4001 33FF	SPI1		Section 25.5 on page 742
0x4001 2C00 - 0x4001 2FFF	TIM1 timer		Section 14.4.21 on page 363
0x4001 2800 - 0x4001 2BFF	ADC2		Section 11.12.15 on page 252
0x4001 2400 - 0x4001 27FF	ADC1		
0x4001 2000 - 0x4001 23FF	GPIO Port G		Section 9.5 on page 194
0x4001 1C00 - 0x4001 1FFF	GPIO Port F		
0x4001 1800 - 0x4001 1BFF	GPIO Port E		
0x4001 1400 - 0x4001 17FF	GPIO Port D		
0x4001 1000 - 0x4001 13FF	GPIO Port C		
0x4001 0C00 - 0x4001 0FFF	GPIO Port B		
0x4001 0800 - 0x4001 0BFF	GPIO Port A		
0x4001 0400 - 0x4001 07FF	EXTI		Section 10.3.7 on page 214
0x4001 0000 - 0x4001 03FF	AFIO		Section 9.5 on page 194

9.5 GPIO and AFIO register maps

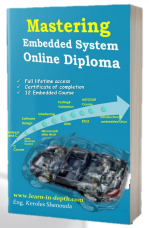
The following tables give the GPIO and AFIO register map and the reset values.

Refer to *Table 3 on page 50* for the register boundary addresses.

Table 59. GPIO register map and reset values

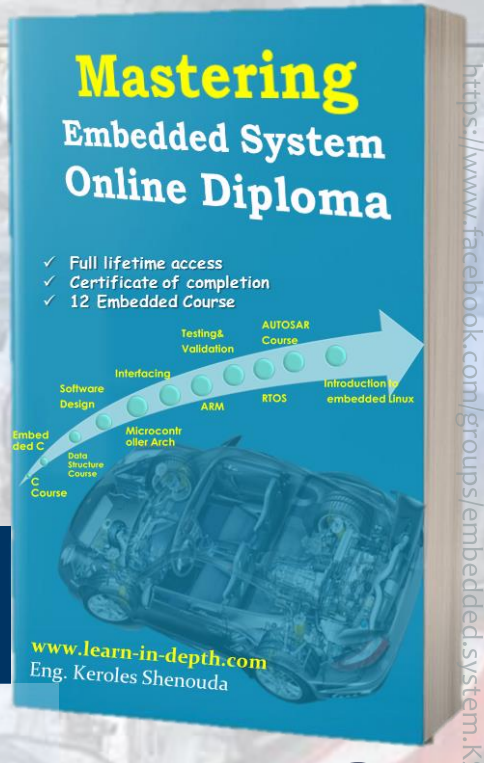
Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	GPIOx_CRL	CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]		CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
	Reset value	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
0x04	GPIOx_CRH	CNF15[1:0]		MODE15[1:0]		CNF14[1:0]		MODE14[1:0]		CNF13[1:0]		MODE13[1:0]		CNF12[1:0]		MODE12[1:0]		CNF11[1:0]		MODE11[1:0]		CNF10[1:0]		MODE10[1:0]		CNF9[1:0]		MODE9[1:0]		CNF8[1:0]		MODE8[1:0]	
	Reset value	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
0x08	GPIOx_IDR	Reserved																IDRy															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	GPIOx_ODR	Reserved																ODRy															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	GPIOx_BSRR	BR[15:0]																BSR[15:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x14	GPIOx_BRR	Reserved																BR[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x18	GPIOx_LCKR	Reserved															LCKK	LCK[15:0]															
	Reset value																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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



#LEARN_IN DEPTH

#Be_professional_in_embedded_system



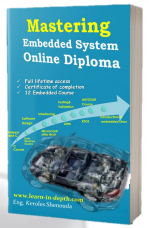
Navigate “Memory map” for TM4C123 in TRM

TRM: TECHNICAL REFERENCE MANUAL

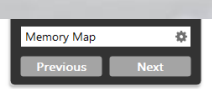
LEARN-IN-DEPTH

Be professional in embedded system

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



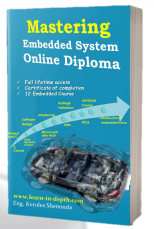
Navigate “Memory map” for TM4C123 in TRM



TEXAS INSTRUMENTS

Tiva™ TM4C123GH6PM Microcontroller

DATA SHEET

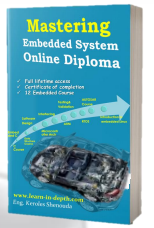


#LEARN_IN DEPTH
#Be_professional_in_embedded_system

Navigate “Memory map” for TM4C123 in TRM

List of Tables

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



Navigate “Memory map” for TM4C123 in TRM

2.4 Memory Model

This section describes the processor memory map, the behavior of memory accesses, and the bit-banding features. The processor has a fixed memory map that provides up to 4 GB of addressable memory.

The memory map for the TM4C123GH6PM controller is provided in Table 2-4 on page 92. In this manual, register addresses are given as a hexadecimal increment, relative to the module's base address as shown in the memory map.

The regions for SRAM and peripherals include bit-band regions. Bit-banding provides atomic operations to bit data (see “Bit-Banding” on page 97).

The processor reserves regions of the Private peripheral bus (PPB) address range for core peripheral registers (see “Cortex-M4 Peripherals” on page 122).

Note: Within the memory map, attempts to read or write addresses in reserved spaces result in a bus fault. In addition, attempts to write addresses in the flash range also result in a bus fault.

Table 2-4. Memory Map

Start	End	Description	For details, see page ...
Memory			
0x0000.0000	0x0003.FFFF	On-chip Flash	540
0x0004.0000	0x1FFF.FFFF	Reserved	-
0x2000.0000	0x2000.7FFF	Bit-banded on-chip SRAM	525
0x2000.8000	0x21FF.FFFF	Reserved	-
0x2200.0000	0x220F.FFFF	Bit-band alias of bit-banded on-chip SRAM starting at 0x2000.0000	525
0x2210.0000	0x3FFF.FFFF	Reserved	-
Peripherals			
0x4000.0000	0x4000.0FFF	Watchdog timer 0	776
0x4000.1000	0x4000.1FFF	Watchdog timer 1	776
0x4000.2000	0x4000.3FFF	Reserved	-
0x4000.4000	0x4000.4FFF	GPIO Port A	658
0x4000.5000	0x4000.5FFF	GPIO Port B	658

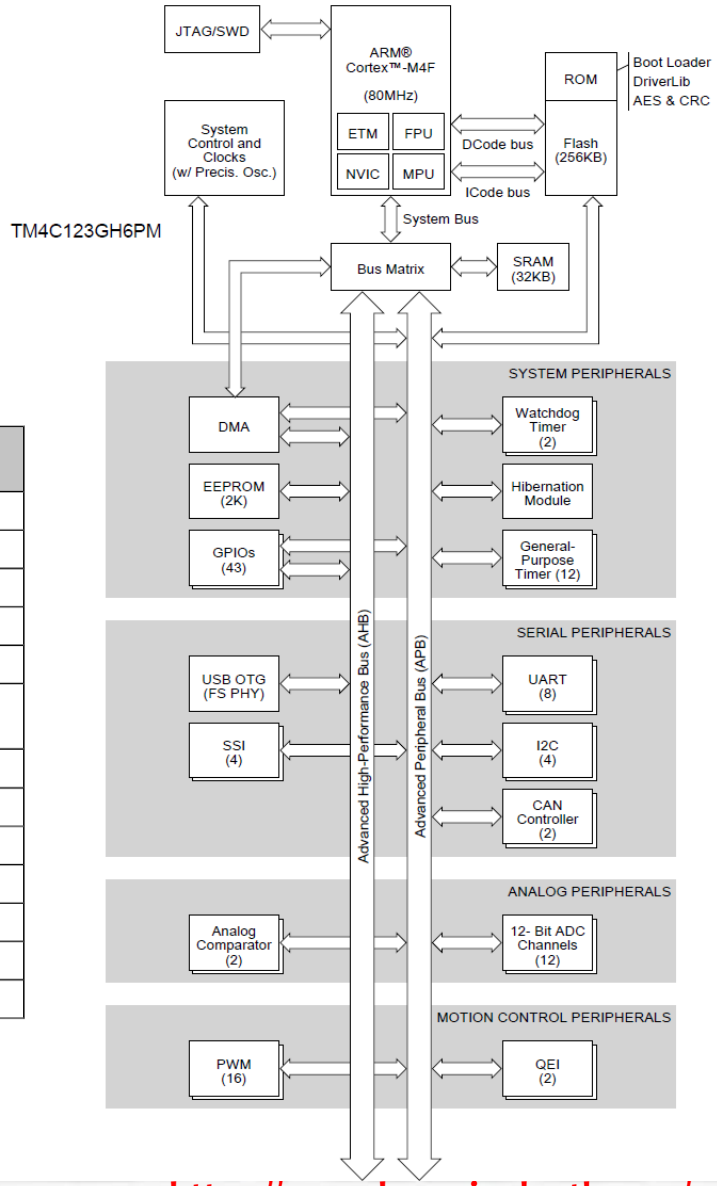
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

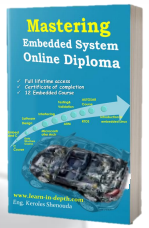
Navigate “Memory map” for TM4C123 in TRM

Table 2-4. Memory Map

Start	End	Description	For details, see page ...
Memory			
0x0000.0000	0x0003.FFFF	On-chip Flash	540
0x0004.0000	0x1FFF.FFFF	Reserved	-
0x2000.0000	0x2000.7FFF	Bit-banded on-chip SRAM	525
0x2000.8000	0x21FF.FFFF	Reserved	-
0x2200.0000	0x220F.FFFF	Bit-band alias of bit-banded on-chip SRAM starting at 0x2000.0000	525
0x2210.0000	0x3FFF.FFFF	Reserved	-
Peripherals			
0x4000.0000	0x4000.0FFF	Watchdog timer 0	776
0x4000.1000	0x4000.1FFF	Watchdog timer 1	776
0x4000.2000	0x4000.3FFF	Reserved	-
0x4000.4000	0x4000.4FFF	GPIO Port A	658
0x4000.5000	0x4000.5FFF	GPIO Port B	658

Figure 1-1. Tiva™ TM4C123GH6PM Microcontroller High-Level Block Diagram





#LEARN_IN DEPTH

#Be_professional_in_embedded_system

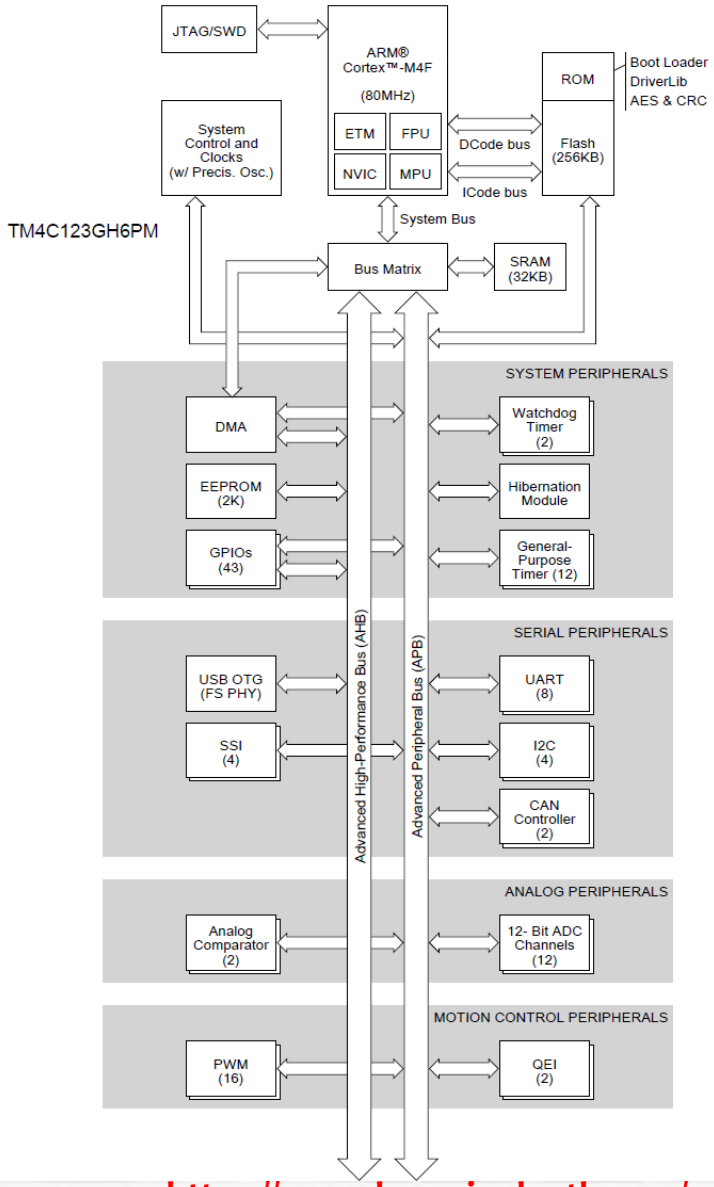
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

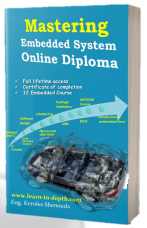
Navigate “Memory map” for TM4C123 in TRM

Table 2-4. Memory Map *(continued)*

Start	End	Description	For details, see page ...
0x4000.6000	0x4000.6FFF	GPIO Port C	658
0x4000.7000	0x4000.7FFF	GPIO Port D	658
0x4000.8000	0x4000.8FFF	SSI0	967
0x4000.9000	0x4000.9FFF	SSI1	967
0x4000.A000	0x4000.AFFF	SSI2	967
0x4000.B000	0x4000.BFFF	SSI3	967
0x4000.C000	0x4000.CFFF	UART0	903
0x4000.D000	0x4000.DFFF	UART1	903
0x4000.E000	0x4000.EFFF	UART2	903
0x4000.F000	0x4000.FFFF	UART3	903
0x4001.0000	0x4001.0FFF	UART4	903
0x4001.1000	0x4001.1FFF	UART5	903
0x4001.2000	0x4001.2FFF	UART6	903
0x4001.3000	0x4001.3FFF	UART7	903
0x4001.4000	0x4001.FFFF	Reserved	-
Peripherals			
0x4002.0000	0x4002.0FFF	I^2C 0	1017
0x4002.1000	0x4002.1FFF	I^2C 1	1017
0x4002.2000	0x4002.2FFF	I^2C 2	1017
0x4002.3000	0x4002.3FFF	I^2C 3	1017
0x4002.4000	0x4002.4FFF	GPIO Port E	658
0x4002.5000	0x4002.5FFF	GPIO Port F	658
0x4002.6000	0x4002.7FFF	Reserved	-
0x4002.8000	0x4002.8FFF	PWM 0	1240
0x4002.9000	0x4002.9FFF	PWM 1	1240
0x4002.A000	0x4002.BFFF	Reserved	-
0x4002.C000	0x4002.CFFF	QEI0	1310
0x4002.D000	0x4002.DFFF	QEI1	1310
0x4002.E000	0x4002.FFFF	Reserved	-
0x4003.0000	0x4003.0FFF	16/32-bit Timer 0	725
0x4003.1000	0x4003.1FFF	16/32-bit Timer 1	725
0x4003.2000	0x4003.2FFF	16/32-bit Timer 2	725
0x4003.3000	0x4003.3FFF	16/32-bit Timer 3	725
0x4003.4000	0x4003.4FFF	16/32-bit Timer 4	725
0x4003.5000	0x4003.5FFF	16/32-bit Timer 5	725
0x4003.6000	0x4003.6FFF	32/64-bit Timer 0	725
0x4003.7000	0x4003.7FFF	32/64-bit Timer 1	725
0x4003.8000	0x4003.8FFF	ADC0	818
0x4003.9000	0x4003.9FFF	ADC1	818
0x4003.A000	0x4003.BFFF	Reserved	-
0x4003.C000	0x4003.CFFF	Analog Comparators	1220

Figure 1-1. Tiva™ TM4C123GH6PM Microcontroller High-Level Block Diagram





#LEARN_IN DEPTH

#Be_professional_in_embedded_system

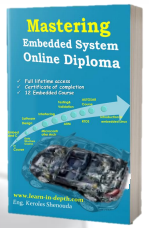
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Offset for each GPIO A, B, C, ...

Table 10-6. GPIO Register Map

Offset	Name	Type	Reset	Description	See page
0x000	GPIODATA	RW	0x0000.0000	GPIO Data	662
0x400	GPIODIR	RW	0x0000.0000	GPIO Direction	663
0x404	GPIOIS	RW	0x0000.0000	GPIO Interrupt Sense	664
0x408	GPIOIBE	RW	0x0000.0000	GPIO Interrupt Both Edges	665
0x40C	GPIOIEV	RW	0x0000.0000	GPIO Interrupt Event	666
0x410	GPIOIM	RW	0x0000.0000	GPIO Interrupt Mask	667
0x414	GPIORIS	RO	0x0000.0000	GPIO Raw Interrupt Status	668
0x418	GPIOMIS	RO	0x0000.0000	GPIO Masked Interrupt Status	669
0x41C	GPIOICR	W1C	0x0000.0000	GPIO Interrupt Clear	670
0x420	GPIOAFSEL	RW	-	GPIO Alternate Function Select	671
0x500	GPIODR2R	RW	0x0000.00FF	GPIO 2-mA Drive Select	673
0x504	GPIODR4R	RW	0x0000.0000	GPIO 4-mA Drive Select	674
0x508	GPIODR8R	RW	0x0000.0000	GPIO 8-mA Drive Select	675
0x50C	GPIOODR	RW	0x0000.0000	GPIO Open Drain Select	676
0x510	GPIOPUR	RW	-	GPIO Pull-Up Select	677
0x514	GPIOPDR	RW	0x0000.0000	GPIO Pull-Down Select	679
0x518	GPIOSLR	RW	0x0000.0000	GPIO Slew Rate Control Select	681
0x51C	GPIODEN	RW	-	GPIO Digital Enable	682
0x520	GPIOLOCK	RW	0x0000.0001	GPIO Lock	684
0x524	GPIOCR	-	-	GPIO Commit	685
0x528	GPIOAMSEL	RW	0x0000.0000	GPIO Analog Mode Select	687
0x52C	GPIOPCTL	RW	-	GPIO Port Control	688
0x530	GPIOADCCTL	RW	0x0000.0000	GPIO ADC Control	690
0x534	GPIODMACTL	RW	0x0000.0000	GPIO DMA Control	691
0xFD0	GPIOPeriphID4	RO	0x0000.0000	GPIO Peripheral Identification 4	692
0xFD4	GPIOPeriphID5	RO	0x0000.0000	GPIO Peripheral Identification 5	693
0xFD8	GPIOPeriphID6	RO	0x0000.0000	GPIO Peripheral Identification 6	694
0xFDC	GPIOPeriphID7	RO	0x0000.0000	GPIO Peripheral Identification 7	695
0xFE0	GPIOPeriphID0	RO	0x0000.0061	GPIO Peripheral Identification 0	696
0xFE4	GPIOPeriphID1	RO	0x0000.0000	GPIO Peripheral Identification 1	697
0xFE8	GPIOPeriphID2	RO	0x0000.0018	GPIO Peripheral Identification 2	698
0xFEC	GPIOPeriphID3	RO	0x0000.0001	GPIO Peripheral Identification 3	699

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



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

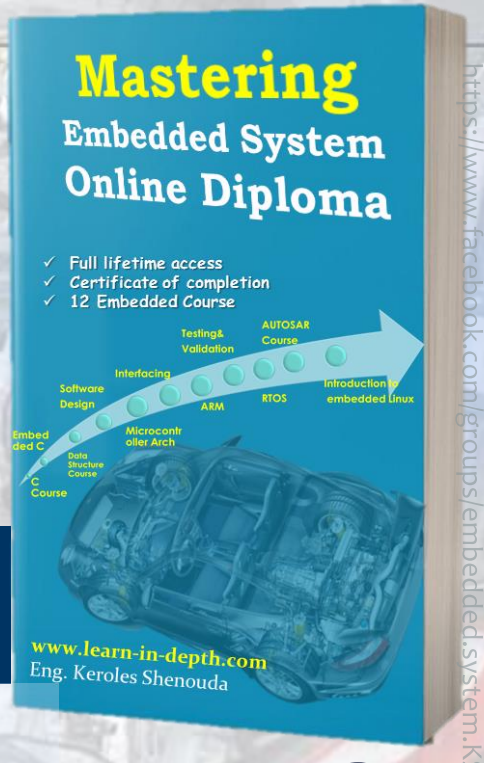
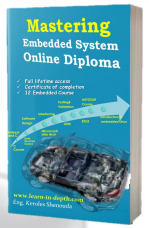


Figure out different based addresses from different MCUs

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

In STM32F103xx

- What is the base address of APB1 BUS peripheral register ?

Figure 1. System architecture (low-, medium-, XL-density devices)

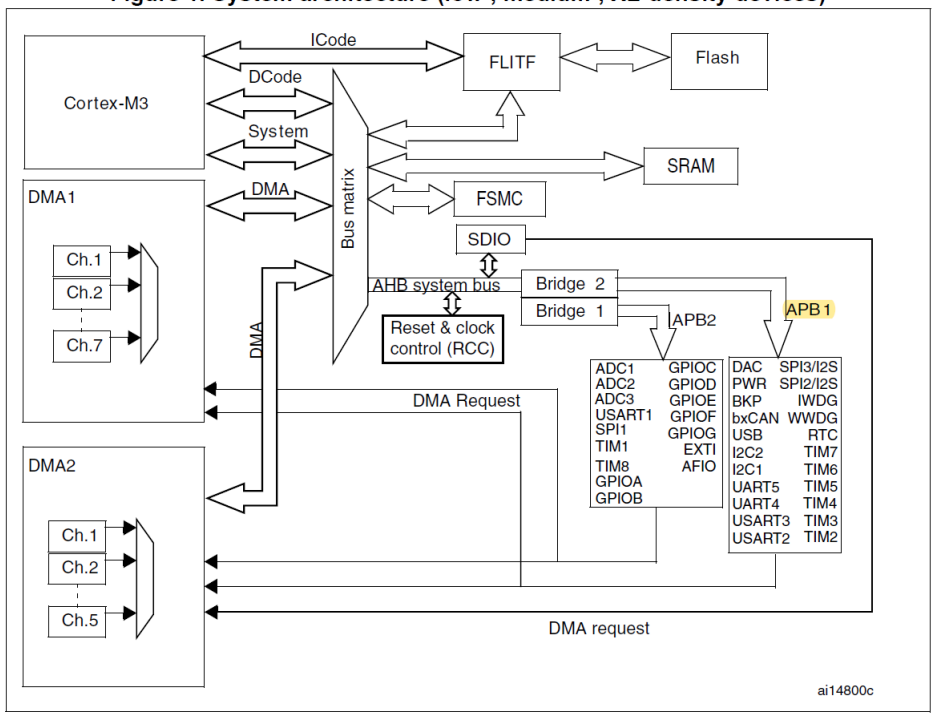
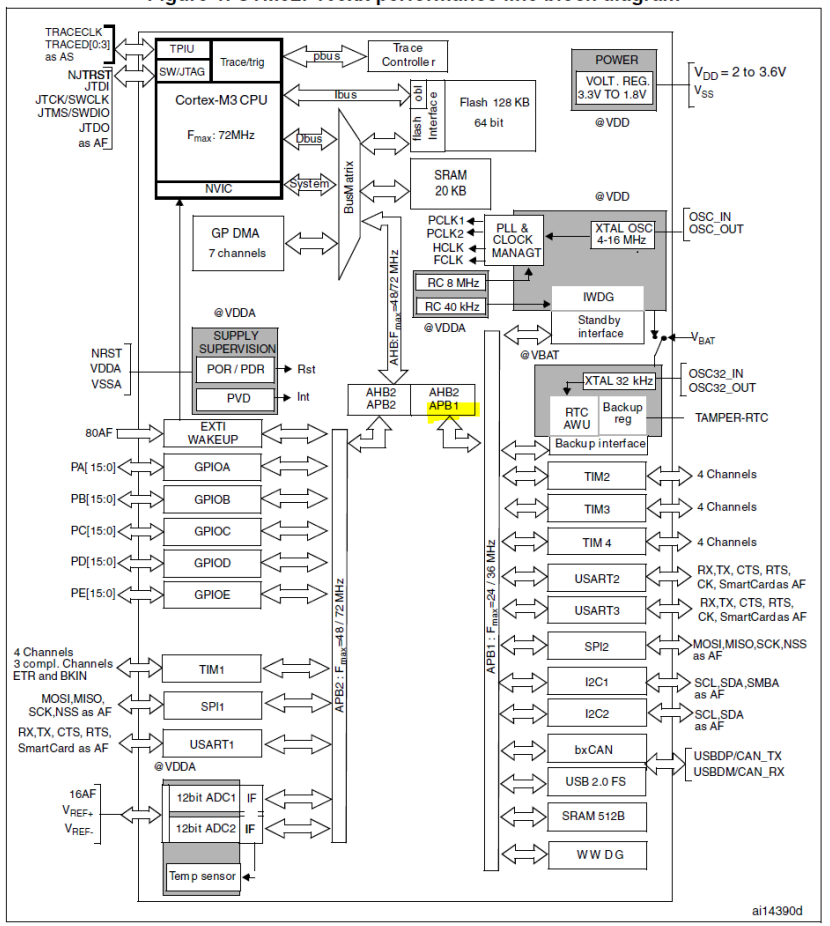


Figure 1. STM32F103xx performance line block diagram



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

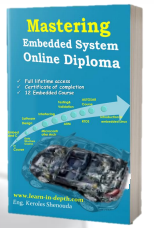
In STM32F103xx

- What is the base address of APB1 BUS peripheral register ?
  - 0x4000_0000

Table 3. Register boundary addresses (continued)

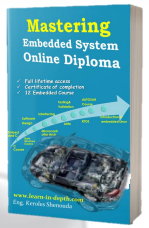
Boundary address	Peripheral	Bus	Register map
0x4000 7800 - 0x4000 FFFF	Reserved	APB1	-
0x4000 7400 - 0x4000 77FF	DAC		Section 12.5.14 on page 273
0x4000 7000 - 0x4000 73FF	Power control PWR		Section 5.4.3 on page 80
0x4000 6C00 - 0x4000 6FFF	Backup registers (BKP)		Section 6.4.5 on page 85
0x4000 6400 - 0x4000 67FF	bxCAN1		Section 24.9.5 on page 695
0x4000 6800 - 0x4000 6BFF	bxCAN2		
0x4000 6000(1) - 0x4000 63FF	Shared USB/CAN SRAM 512 bytes		-
0x4000 5C00 - 0x4000 5FFF	USB device FS registers		Section 23.5.4 on page 651
0x4000 5800 - 0x4000 5BFF	I2C2		Section 26.6.10 on page 784
0x4000 5400 - 0x4000 57FF	I2C1		
0x4000 5000 - 0x4000 53FF	UART5		Section 27.6.8 on page 827
0x4000 4C00 - 0x4000 4FFF	UART4		
0x4000 4800 - 0x4000 4BFF	USART3		
0x4000 4400 - 0x4000 47FF	USART2		
0x4000 4000 - 0x4000 43FF	Reserved		-
0x4000 3C00 - 0x4000 3FFF	SPI3/I2S		Section 25.5 on page 742
0x4000 3800 - 0x4000 3BFF	SPI2/I2S		Section 25.5 on page 742
0x4000 3400 - 0x4000 37FF	Reserved		-
0x4000 3000 - 0x4000 33FF	Independent watchdog (IWDG)		Section 19.4.5 on page 499
0x4000 2C00 - 0x4000 2FFF	Window watchdog (WWDG)		Section 20.6.4 on page 506
0x4000 2800 - 0x4000 2BFF	RTC		Section 18.4.7 on page 493
0x4000 2400 - 0x4000 27FF	Reserved		-
0x4000 2000 - 0x4000 23FF	TIM14 timer		Section 16.5.11 on page 468
0x4000 1C00 - 0x4000 1FFF	TIM13 timer		
0x4000 1800 - 0x4000 1BFF	TIM12 timer		Section 16.4.13 on page 458
0x4000 1400 - 0x4000 17FF	TIM7 timer		Section 17.4.9 on page 481
0x4000 1000 - 0x4000 13FF	TIM6 timer		
0x4000 0C00 - 0x4000 0FFF	TIM5 timer		Section 15.4.19 on page 423
0x4000 0800 - 0x4000 0BFF	TIM4 timer		
0x4000 0400 - 0x4000 07FF	TIM3 timer		
0x4000 0000 - 0x4000 03FF	TIM2 timer		

1. This shared SRAM can be fully accessed only in low-, medium-, high- and XL-density devices, not in connectivity line devices.



In STM32F103xx

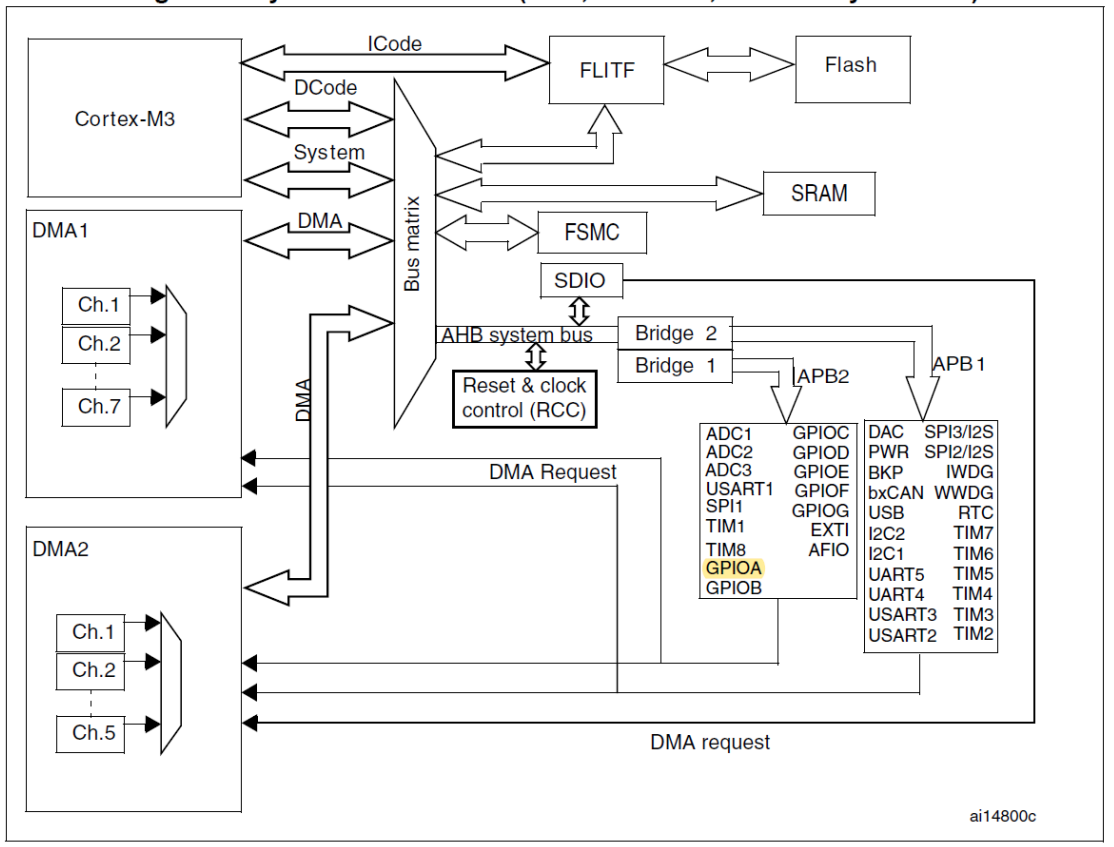
- What is the base address of GPIOA module ?



In STM32F103xx

- What is the base address of GPIOA module ?

Figure 1. System architecture (low-, medium-, XL-density devices)

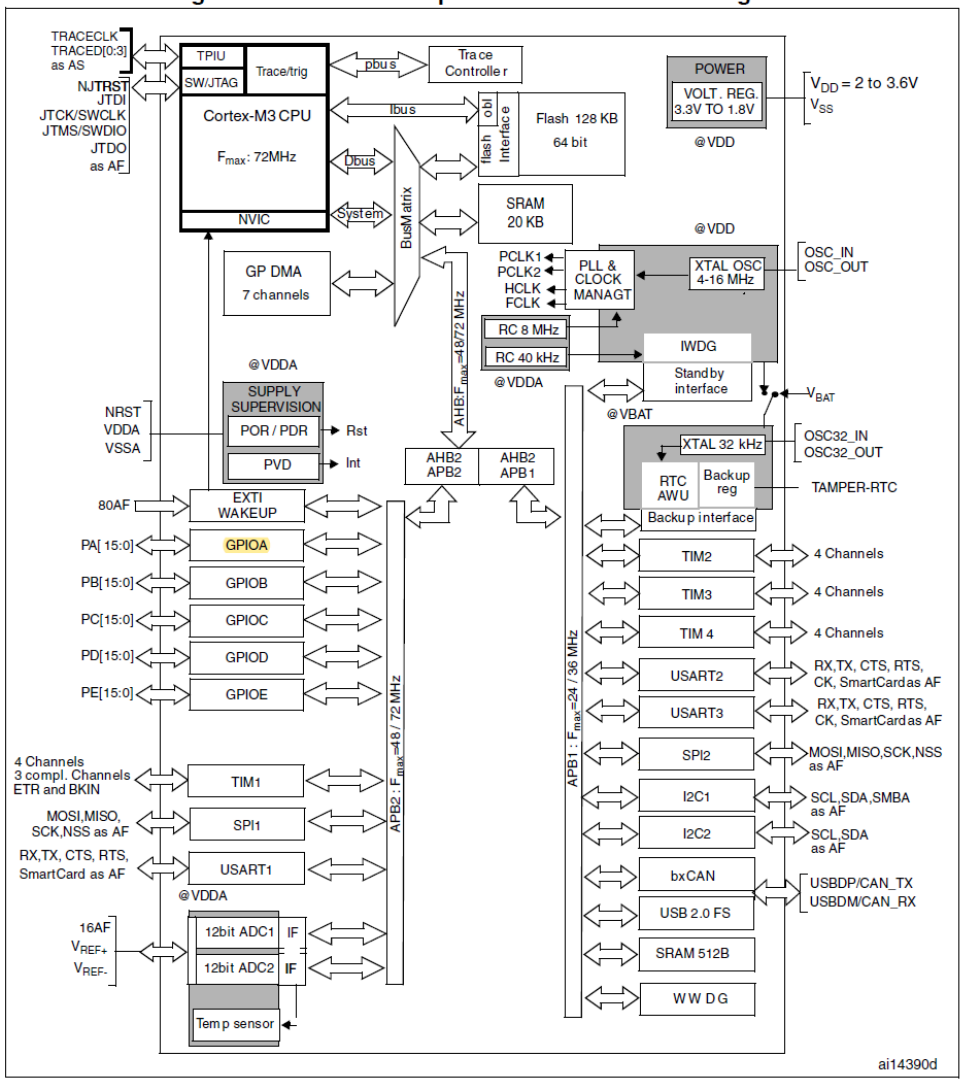


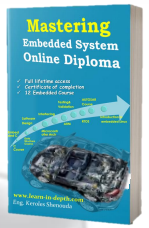
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

In STM32F103xx

- What is the base address of GPIOA module ?

Figure 1. STM32F103xx performance line block diagram





In STM32F103xx

- What is the base address of GPIOA module ?
- 0x4001_0800

Table 3. Register boundary addresses (continued)

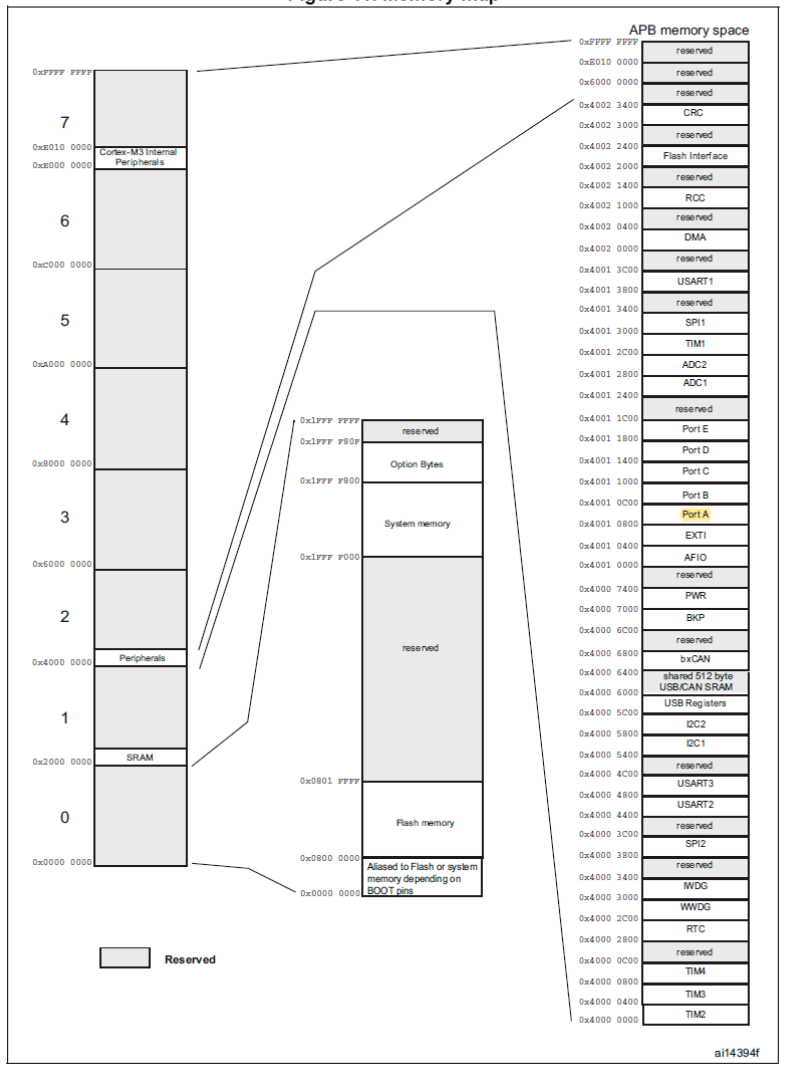
Boundary address	Peripheral	Bus	Register map
0x4001 5800 - 0x4001 7FFF	Reserved	APB2	-
0x4001 5400 - 0x4001 57FF	TIM11 timer		*Section 16.5.11 on page 468*
0x4001 5000 - 0x4001 53FF	TIM10 timer		*Section 16.5.11 on page 468*
0x4001 4C00 - 0x4001 4FFF	TIM9 timer		*Section 16.4.13 on page 458*
0x4001 4000 - 0x4001 4BFF	Reserved		-
0x4001 3C00 - 0x4001 3FFF	ADC3		*Section 11.12.15 on page 252*
0x4001 3800 - 0x4001 3BFF	USART1		*Section 27.6.8 on page 827*
0x4001 3400 - 0x4001 37FF	TIM8 timer		*Section 14.4.21 on page 363*
0x4001 3000 - 0x4001 33FF	SPI1		*Section 25.5 on page 742*
0x4001 2C00 - 0x4001 2FFF	TIM1 timer		*Section 14.4.21 on page 363*
0x4001 2800 - 0x4001 2BFF	ADC2		*Section 11.12.15 on page 252*
0x4001 2400 - 0x4001 27FF	ADC1		
0x4001 2000 - 0x4001 23FF	GPIO Port G		*Section 9.5 on page 194*
0x4001 1C00 - 0x4001 1FFF	GPIO Port F		
0x4001 1800 - 0x4001 1BFF	GPIO Port E		
0x4001 1400 - 0x4001 17FF	GPIO Port D		
0x4001 1000 - 0x4001 13FF	GPIO Port C		
0x4001 0C00 - 0x4001 0FFF	GPIO Port B		
0x4001 0800 - 0x4001 0BFF	GPIO Port A		
0x4001 0400 - 0x4001 07FF	EXTI		*Section 10.3.7 on page 214*
0x4001 0000 - 0x4001 03FF	AFIO		*Section 9.5 on page 194*

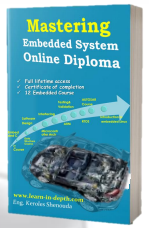
In STM32F103xx

- What is the base address of GPIOA module ?
- 0x4001_0800

TM4C123_MCU.pdf

Figure 11. Memory map

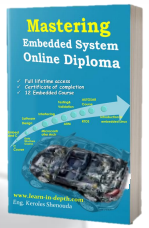




In STM32F103xx

- What is the max frequency for ARM Cortex M3

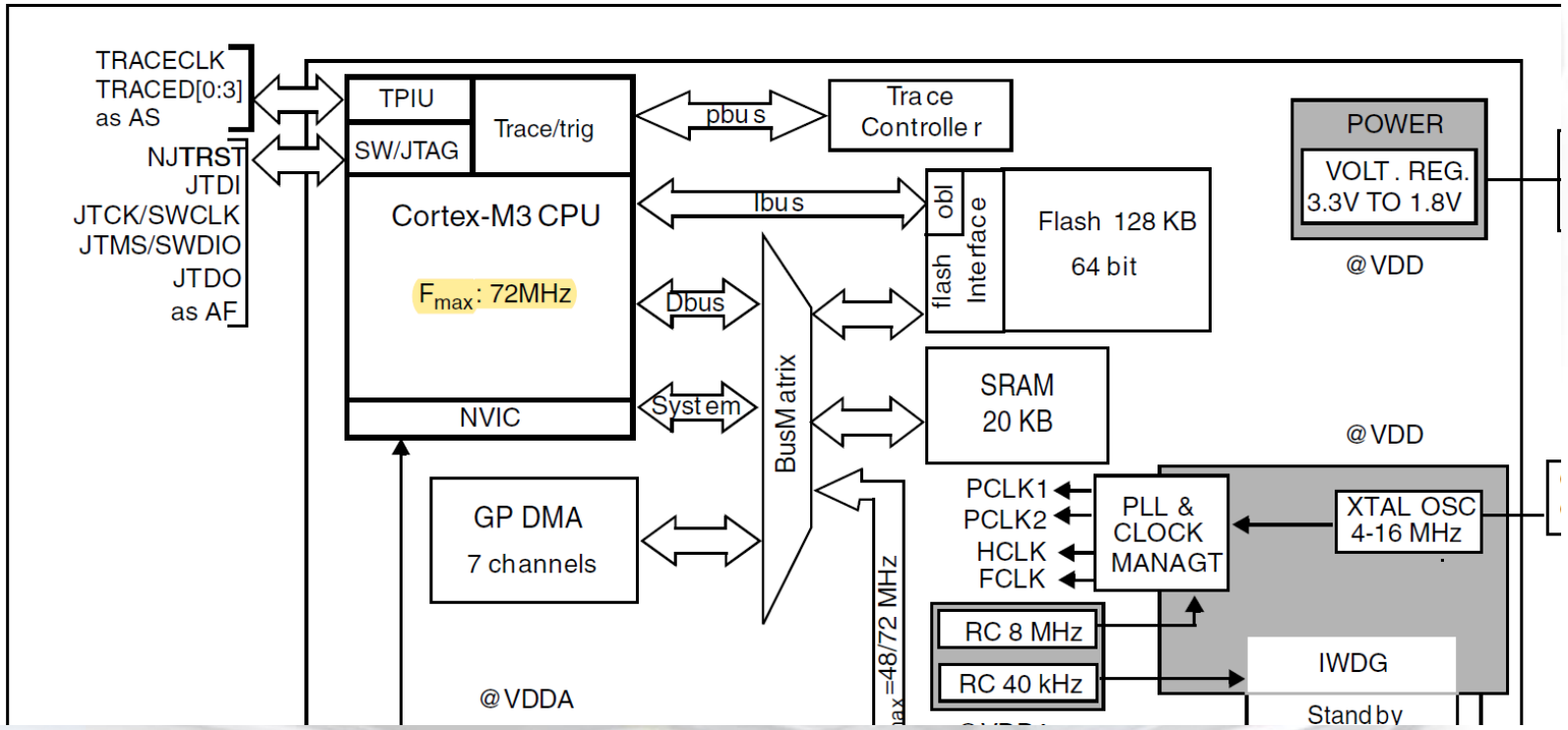
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



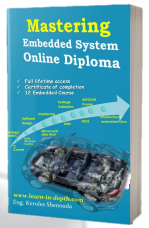
In STM32F103xx

- What is the max frequency for ARM Cortex M3

Figure 1. STM32F103xx performance line block diagram



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



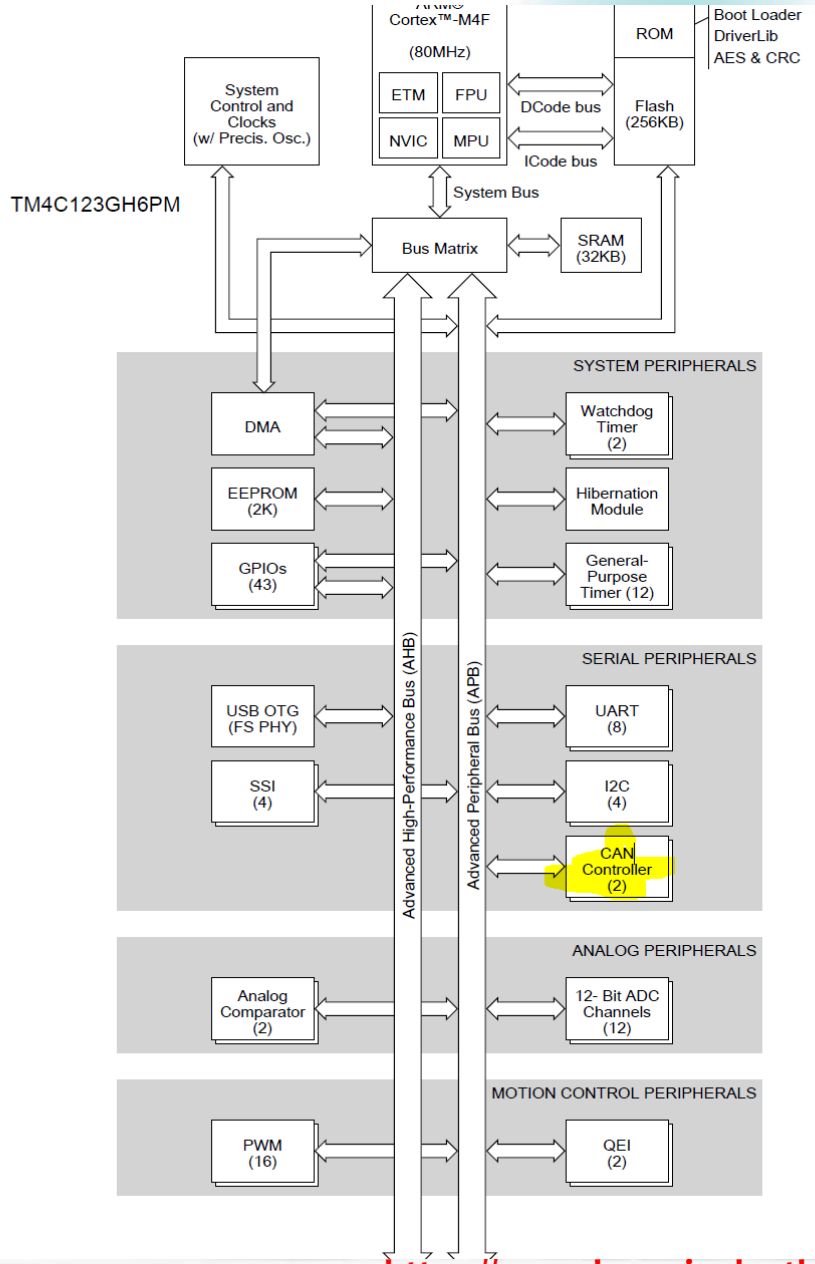
In TM4C123

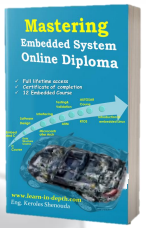
- What is the address for CAN Bit Timing "CANBIT" register in CAN Controller 1

In TM4C123

- What is the address for CAN Bit Timing "CANBIT" register in

CAN Controller 1

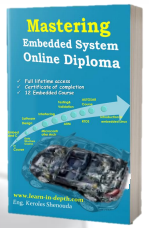




#LEARN_IN DEPTH

#Be_professional_in embedded_system

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



In TM4C123

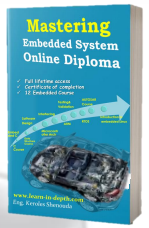
- What is the address for CAN Bit Timing "CANBIT" register in

CAN Controller 1

Base address = 0x4004_1000

Table 2-4. Memory Map *(continued)*

Start	End	Description	For details, see page ...
0x4003.D000	0x4003.FFFF	Reserved	-
0x4004.0000	0x4004.0FFF	CAN0 Controller	1067
0x4004.1000	0x4004.1FFF	CAN1 Controller	1067
0x4004.2000	0x4004.BFFF	Reserved	-
0x4004.C000	0x4004.CFFF	32/64-bit Timer 2	725
0x4004.D000	0x4004.DFFF	32/64-bit Timer 3	725
0x4004.E000	0x4004.EFFF	32/64-bit Timer 4	725
0x4004.F000	0x4004.FFFF	32/64-bit Timer 5	725
0x4005.0000	0x4005.0FFF	USB	1114
0x4005.1000	0x4005.7FFF	Reserved	-
0x4005.8000	0x4005.8FFF	GPIO Port A (AHB aperture)	658
0x4005.9000	0x4005.9FFF	GPIO Port B (AHB aperture)	658
0x4005.A000	0x4005.AFFF	GPIO Port C (AHB aperture)	658
0x4005.B000	0x4005.BFFF	GPIO Port D (AHB aperture)	658
0x4005.C000	0x4005.CFFF	GPIO Port E (AHB aperture)	658
0x4005.D000	0x4005.DFFF	GPIO Port F (AHB aperture)	658



In TM4C123

- What is the address for CAN Bit Timing "CANBIT" register in

CAN Controller 1

Base address = 0x4004_1000

Offset = 0x000C

CANBIT_address =

0x4004_100C

17.4 Register Map

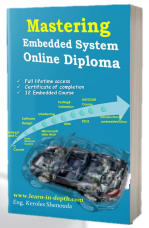
Table 17-5 on page 1067 lists the registers. All addresses given are relative to the CAN base address of:

- CAN0: 0x4004.0000
- CAN1: 0x4004.1000

Note that the CAN controller clock must be enabled before the registers can be programmed (see page 351). There must be a delay of 3 system clocks after the CAN module clock is enabled before any CAN module registers are accessed.

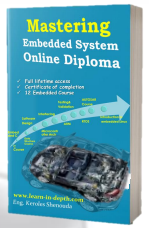
Table 17-5. CAN Register Map

Offset	Name	Type	Reset	Description	See page
0x000	CANCTL	RW	0x0000.0001	CAN Control	1070
0x004	CANSTS	RW	0x0000.0000	CAN Status	1072
0x008	CANERR	RO	0x0000.0000	CAN Error Counter	1075
0x00C	CANBIT	RW	0x0000.2301	CAN Bit Timing	1076
0x010	CANINT	RO	0x0000.0000	CAN Interrupt	1077
0x014	CANTST	RW	0x0000.0000	CAN Test	1078
0x018	CANBRPE	RW	0x0000.0000	CAN Baud Rate Prescaler Extension	1080
0x020	CANIF1CRQ	RW	0x0000.0001	CAN IF1 Command Request	1081



In TM4C123

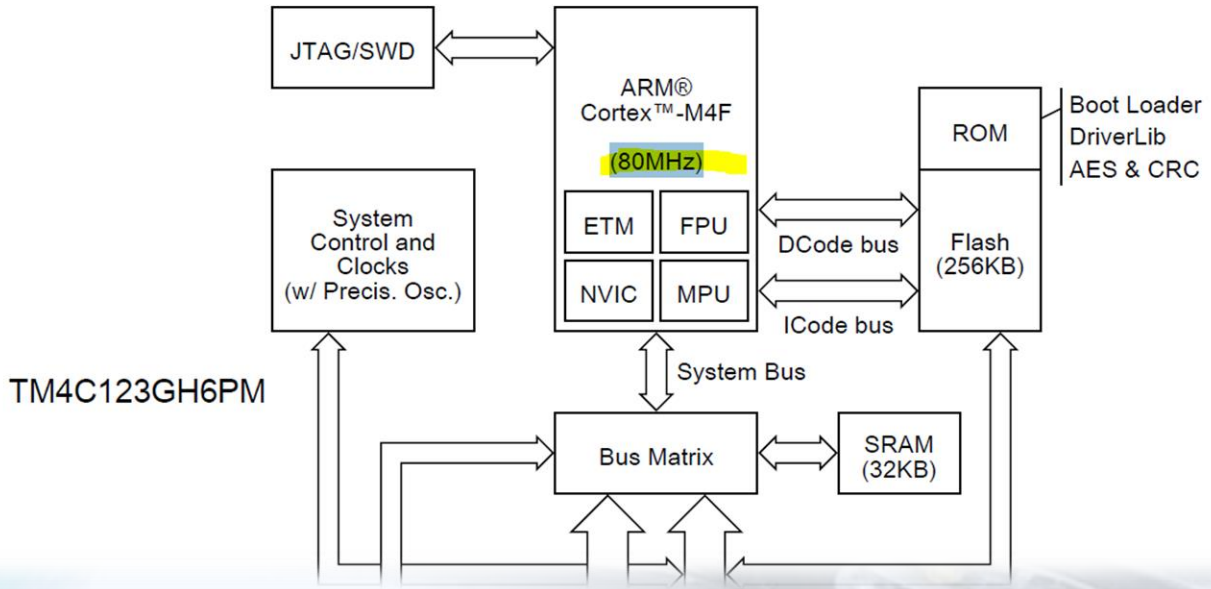
- What is the max frequency for ARM Cortex™-M4 ?



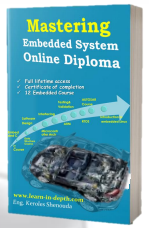
In TM4C123

- What is the max frequency for ARM Cortex™-M4 ?

Figure 1-1. Tiva™ TM4C123GH6PM Microcontroller High-Level Block Diagram



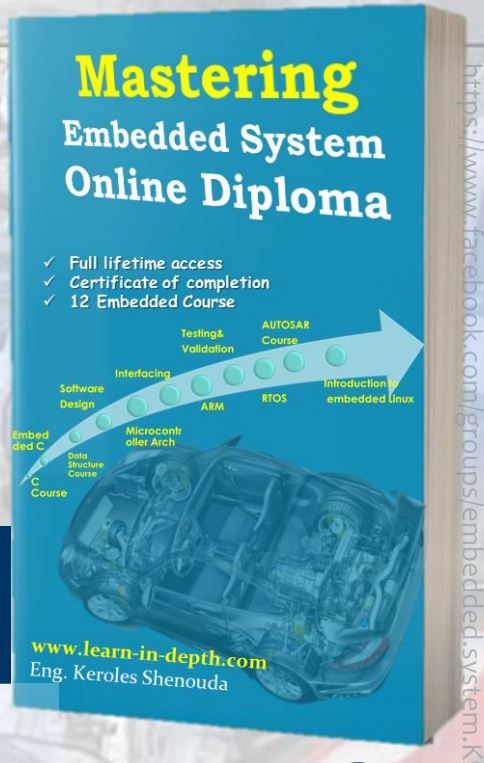
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

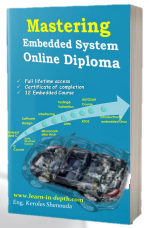
#Be_professional_in_embedded_system

MCU Bus Interfaces



LEARN-IN-DEPTH
Be professional in embedded system

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



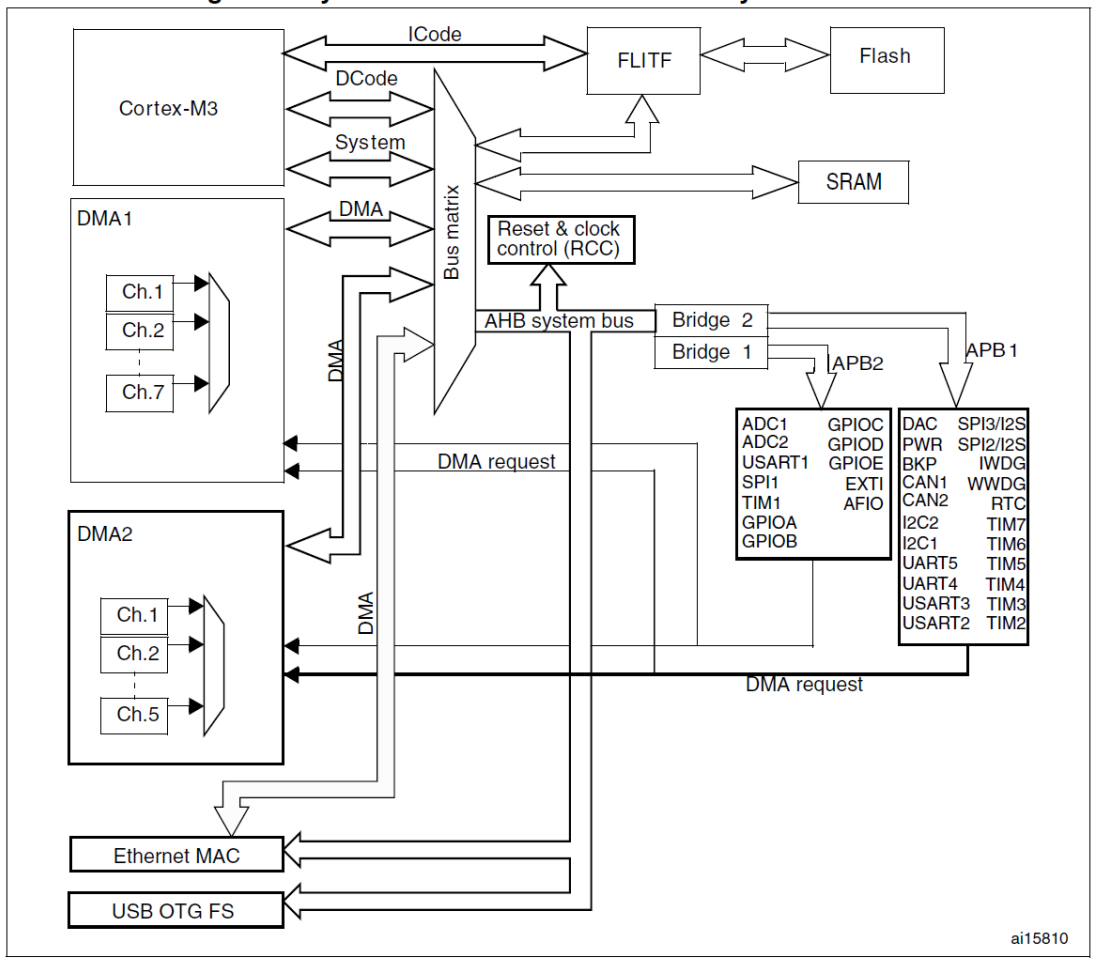
BUS Master/ Slave Concepts

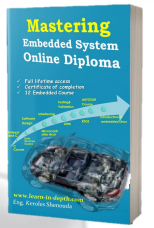
In connectivity line devices the main system consists of:

- Five masters:
  - Cortex®-M3 core DCode bus (D-bus) and System bus (S-bus)
  - GP-DMA1 & 2 (general-purpose DMA)
  - Ethernet DMA
- Three slaves:
  - Internal SRAM
  - Internal Flash memory
  - AHB to APB bridges (AHB to APBx), which connect all the APB peripherals

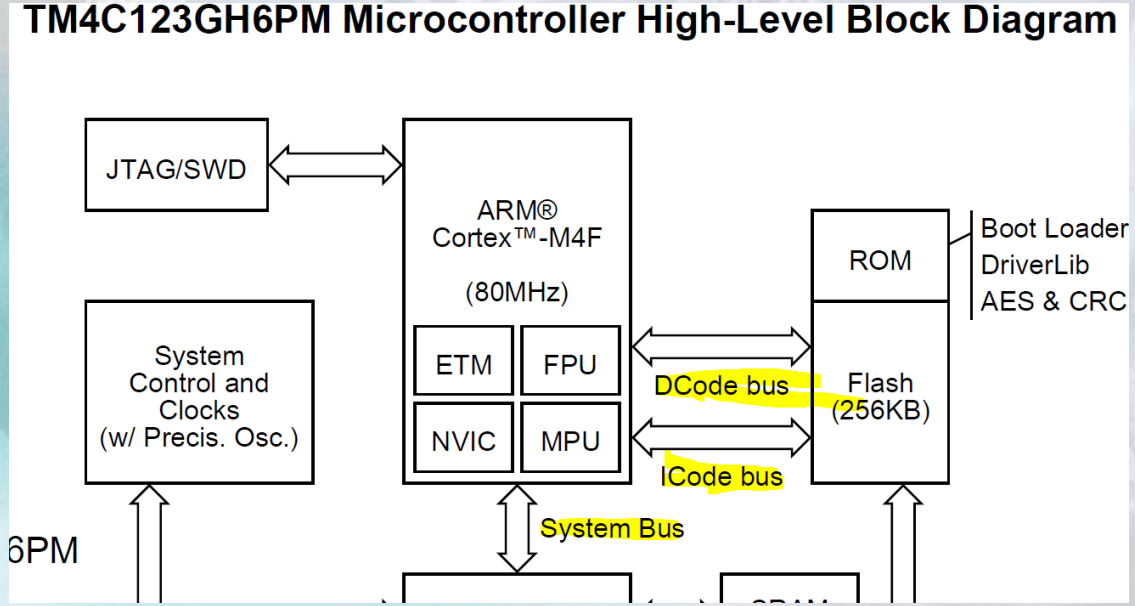
These are interconnected using a multilayer AHB bus architecture as shown in *Figure 2*:

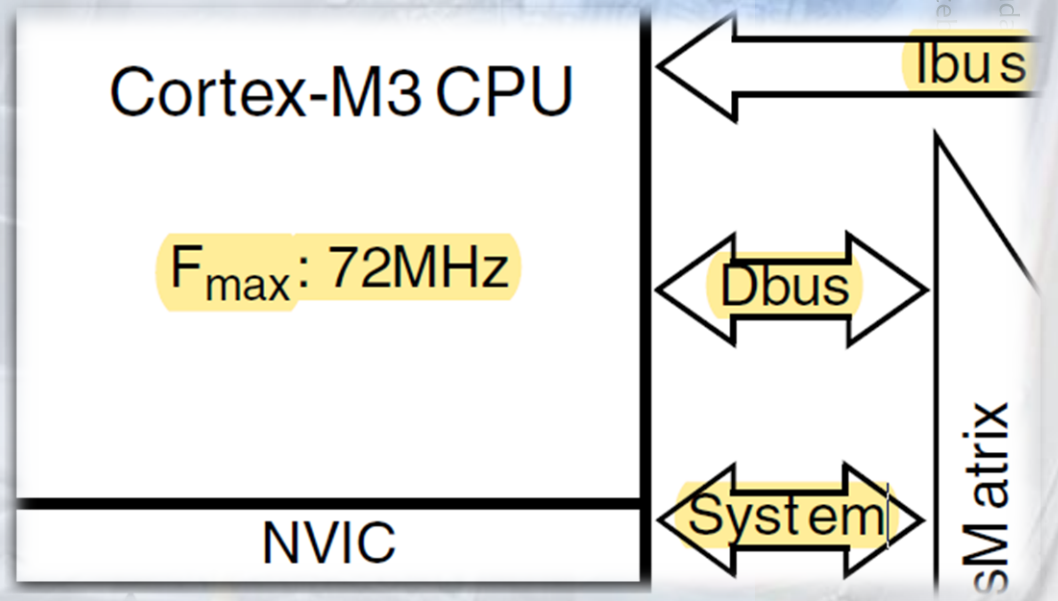
Figure 2. System architecture in connectivity line devices



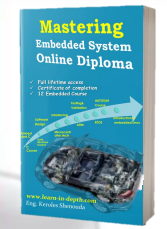


BUS (Ibus / DBUS / System) Bus

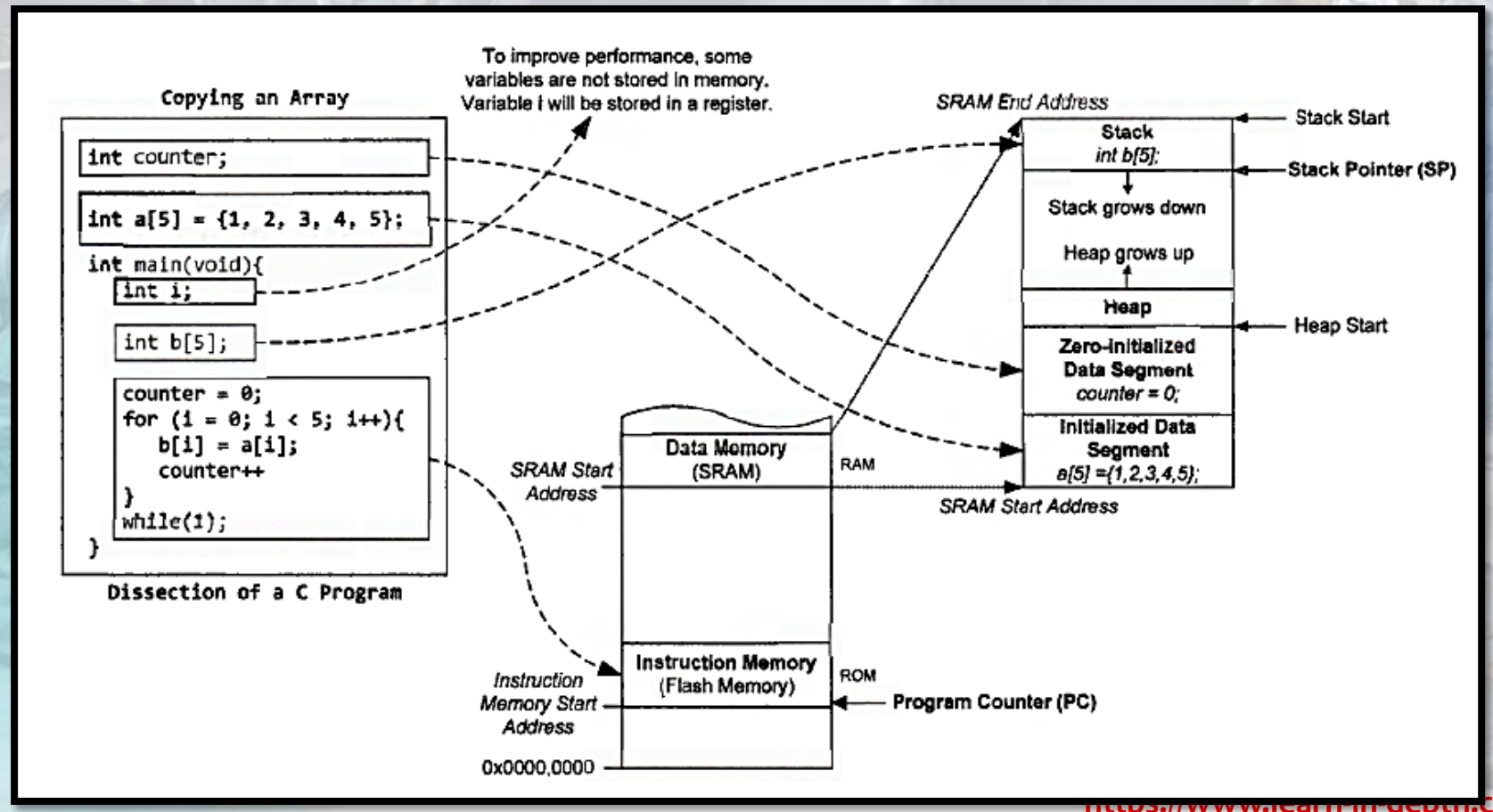




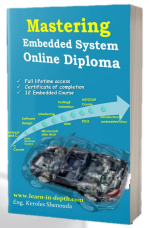
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



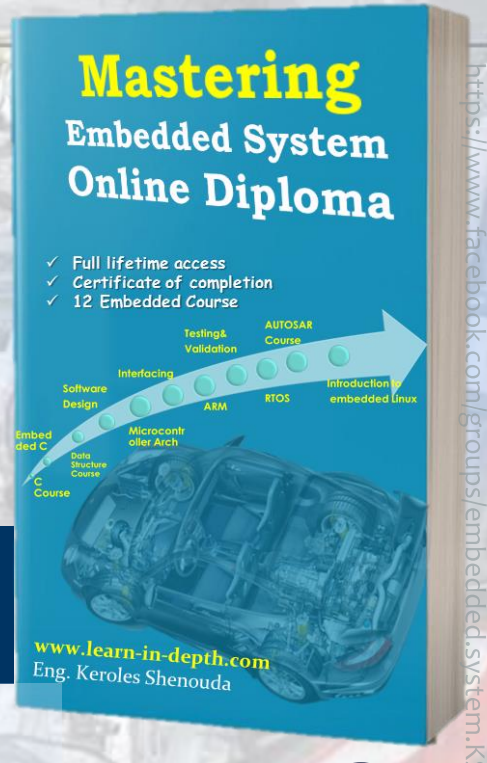
BUS (Ibus / DBUS / System) Bus



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

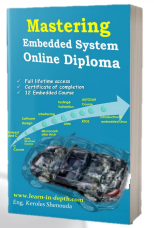


Understanding AMBA Bus Architecture and Protocols



LEARN-IN-DEPTH
Be professional in embedded system

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



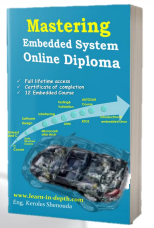
Advanced Micro controller Bus Architecture (AMBA)

- The Advanced Micro controller Bus Architecture (**AMBA**) bus protocols is a set of interconnect specifications from ARM
- that standardizes on chip communication mechanisms between various functional blocks (or IP) for building high performance SOC designs
- These designs typically have one or more micro controllers or microprocessors along with several other components
  - internal memory or external memory bridge, DSP, DMA, accelerators and various other peripherals like USB, UART, PCIE, I2C etc
  - all integrated on a single chip

The ARM Advanced Microcontroller Bus Architecture (AMBA).

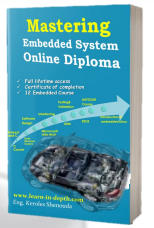
- Open-standard.
- On-chip interconnect specification.
- Defines the connection and management of functional blocks in system on chips (SoCs).

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



Why use AMBA

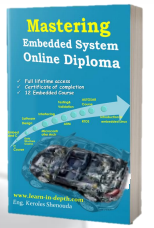
- IP reuse
  - is an essential component in reducing SoC development costs and time period for manfuature.
  - AMBA specifications provide the interface standard that enables IP reuse
  - this is why thousands of SoCs and IP products are using AMBA interfaces
- flexibility
  - AMBA also offers flexibility to work with a range of SoCs



Bus Interface Performance

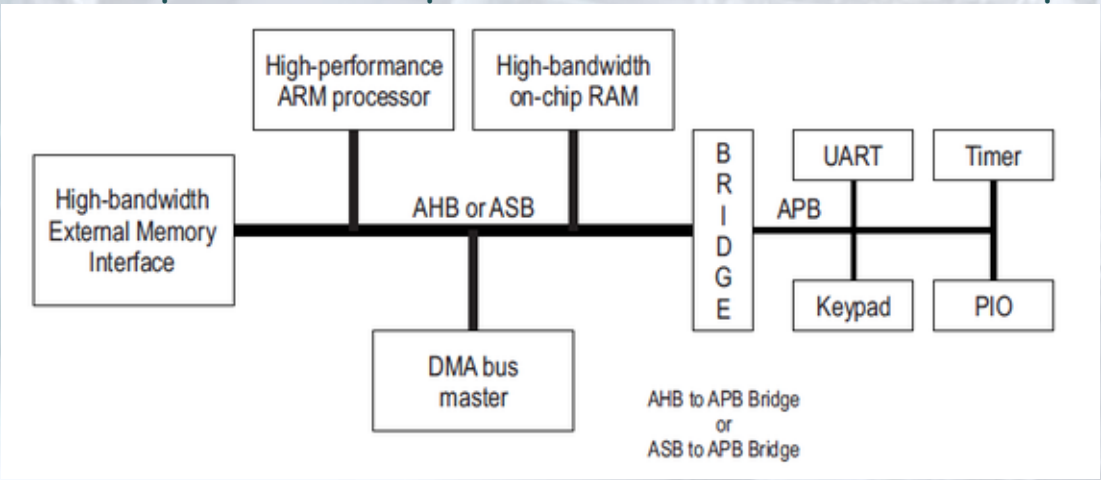
- two main characteristics of **bus interface performance** are **bandwidth** and **latency**
- bandwidth
  - done with this the rate at which data can be driven across the interface in asynchronous system
  - the maximum bandwidth is **limited** by the clock speed and the width of the data bus .

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

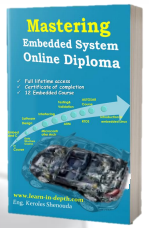


Advanced Micro controller Bus Architecture (AMBA)

- a traditional AMBA based SOC design that uses the
  - AHB (Advanced High performance) or
  - ASB (Advanced System Bus) protocols for high bandwidth interconnect and
  - an APB (Advanced Peripheral Bus) protocol for low bandwidth peripheral interconnects.

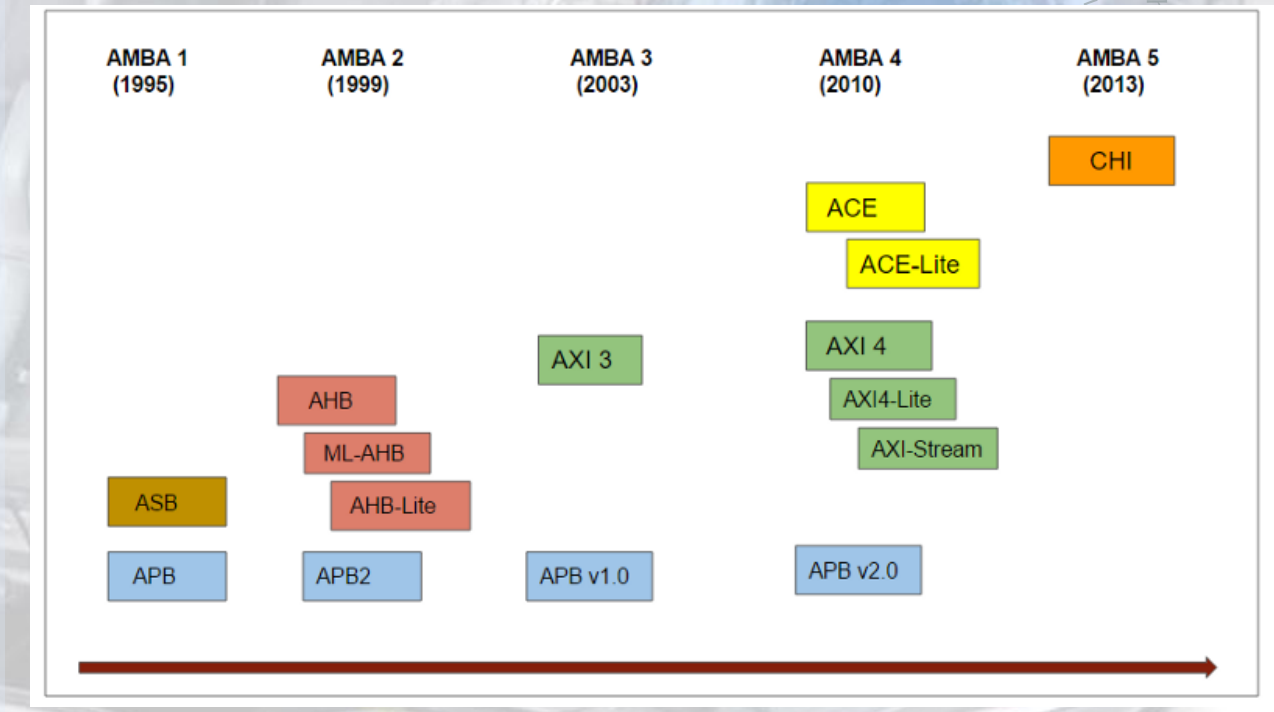


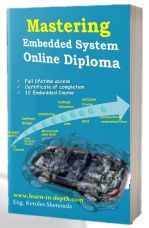
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



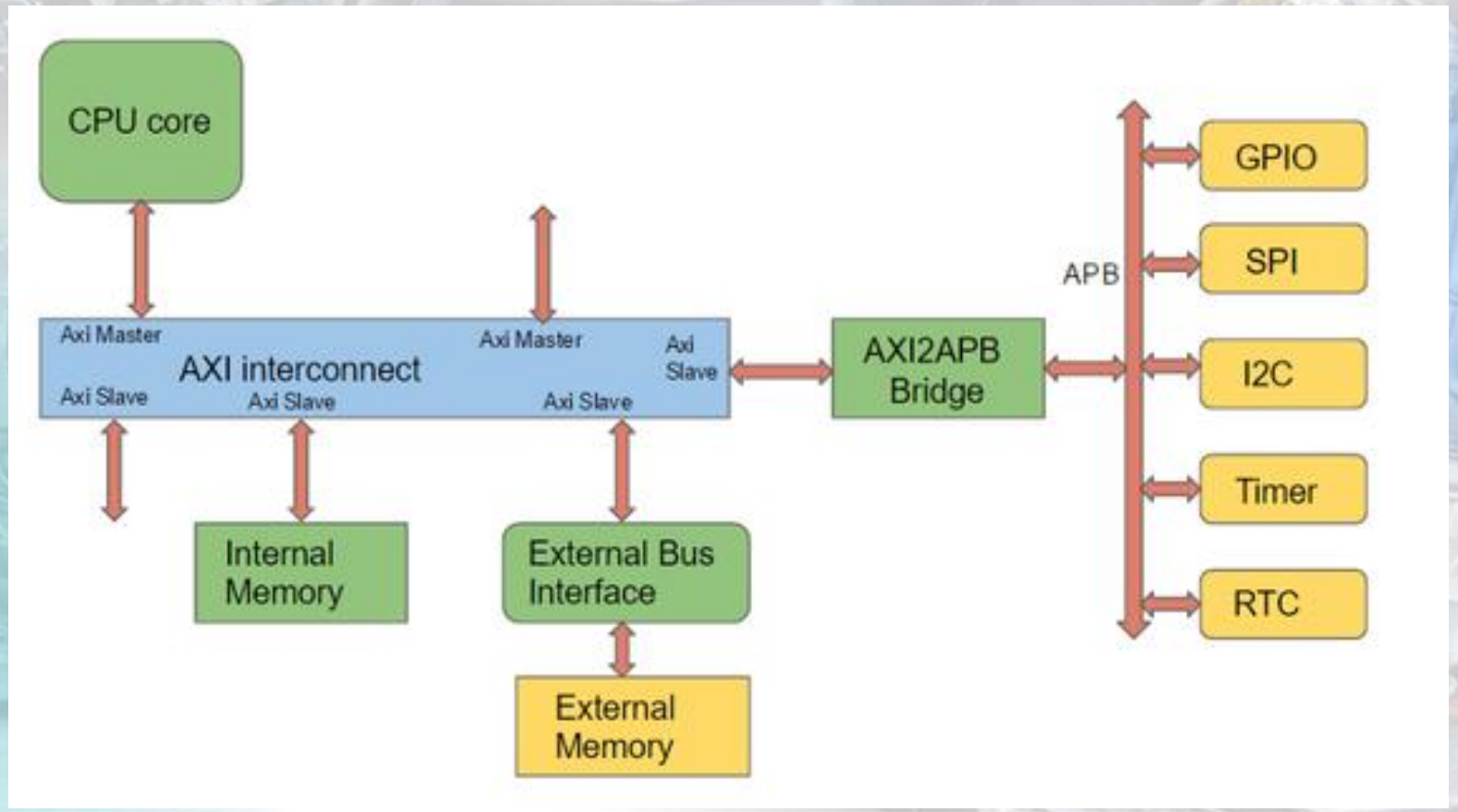
AMBA

- With increasing number of functional blocks (IP) integrating into SOC designs, the shared bus protocols (AHB/ASB) started hitting limitations sooner and in 2003 , the new revision of AMBA 3 introduced a point to point connectivity protocol — AXI (Advanced Extensible Interface). Further in 2010, an enhanced version was introduced — AXI 4. Following diagram illustrates this evolution of protocols along with the SOC design trends in industry

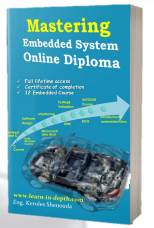




AXI (Advanced Extensible Interface)



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

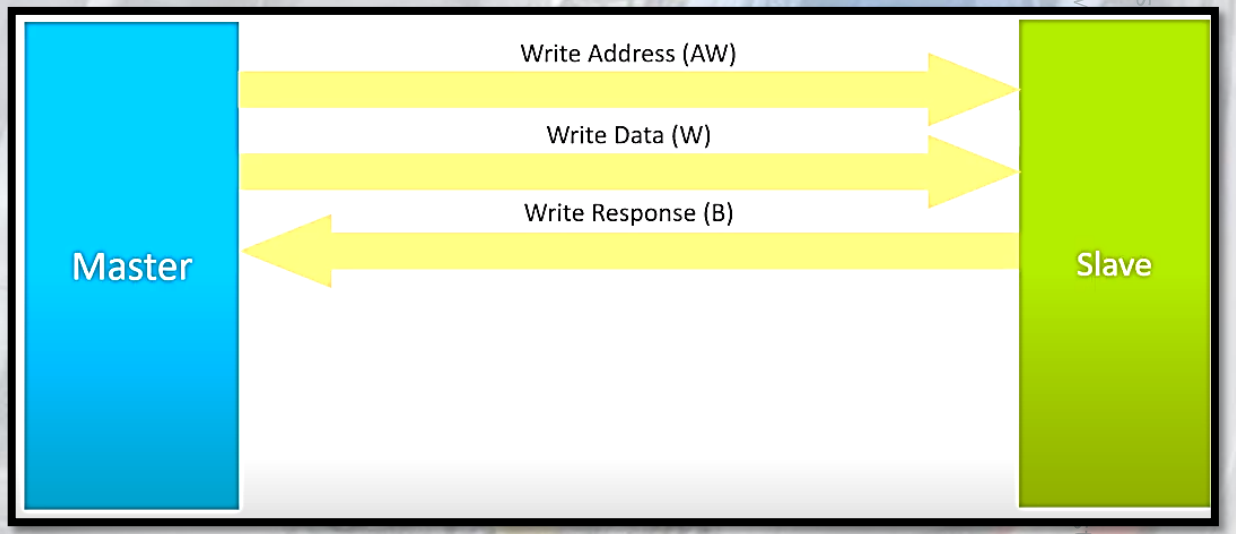


AXI Channels

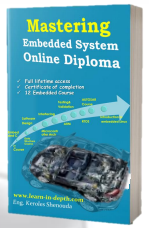
- write address Channel
- write Data Channel

from the master to the slave

- masters use the **write address Channel** to send a memory address of where is going to write the data that is transferred using **the Write data channel**
- Write response channel from slave to master
  - Indicate that the slave has completed the write operation

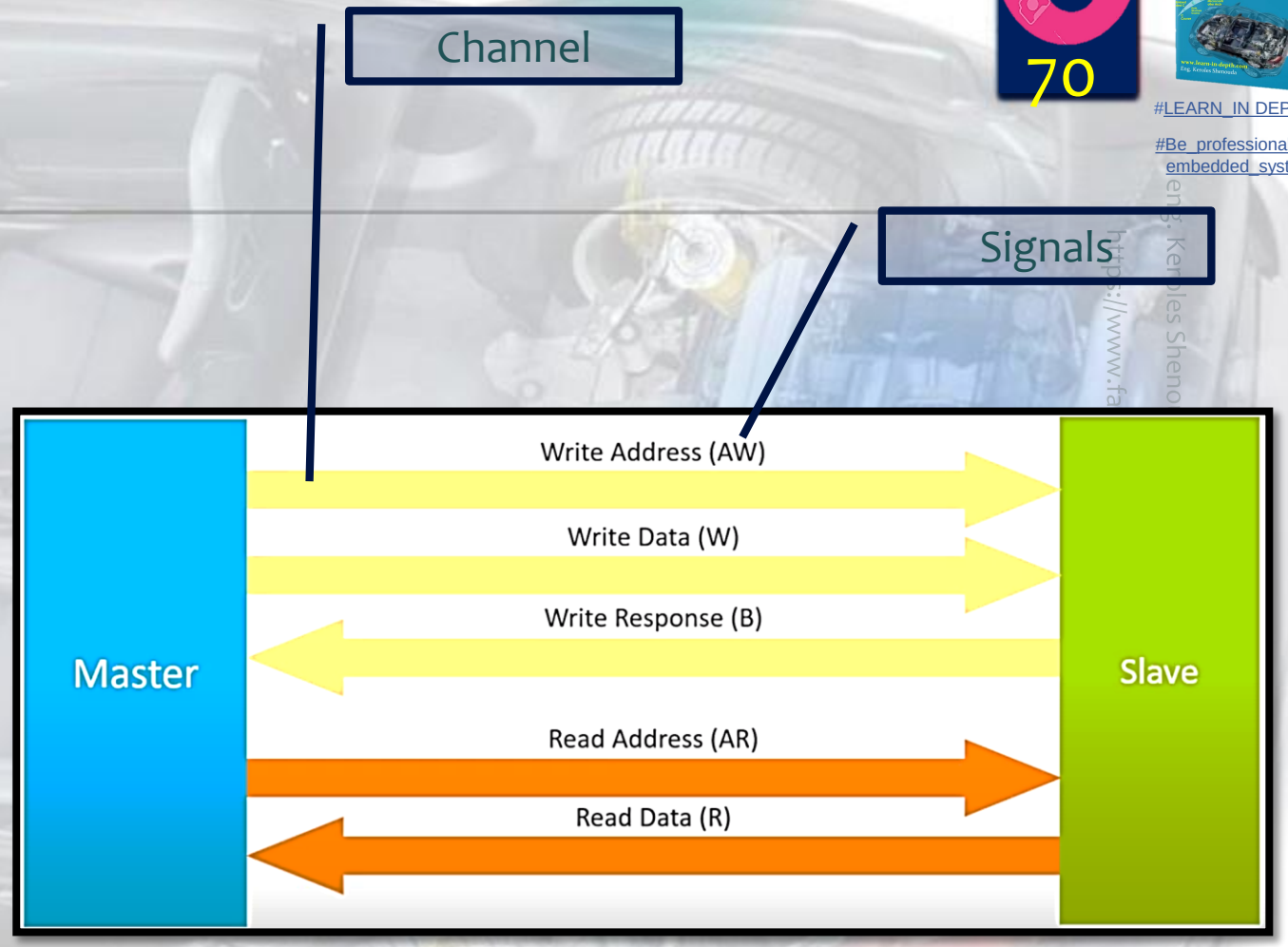


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

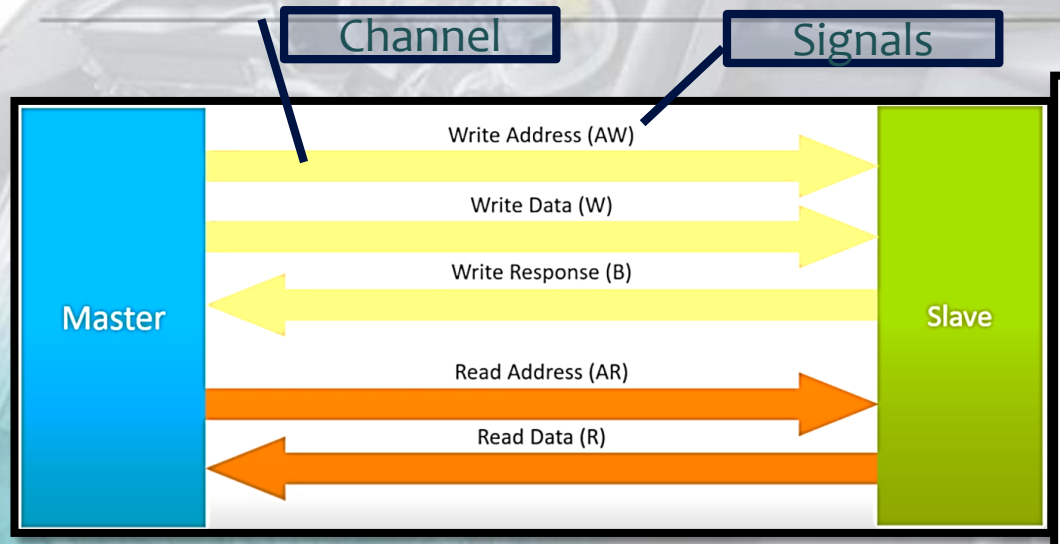


AXI Channels

- read address Channel from the master to the slave
  - the master uses this to specify the address that it will perform a read operation from the slave
- when the date is ready will send the data to the master via the **read data channel**
- if the address **is not valid** or **the data are corrupted or the access doesn't have the right security permission** to slave
  - Slave will return an error message



AXI Channels/Signals



All those signals for only one Port

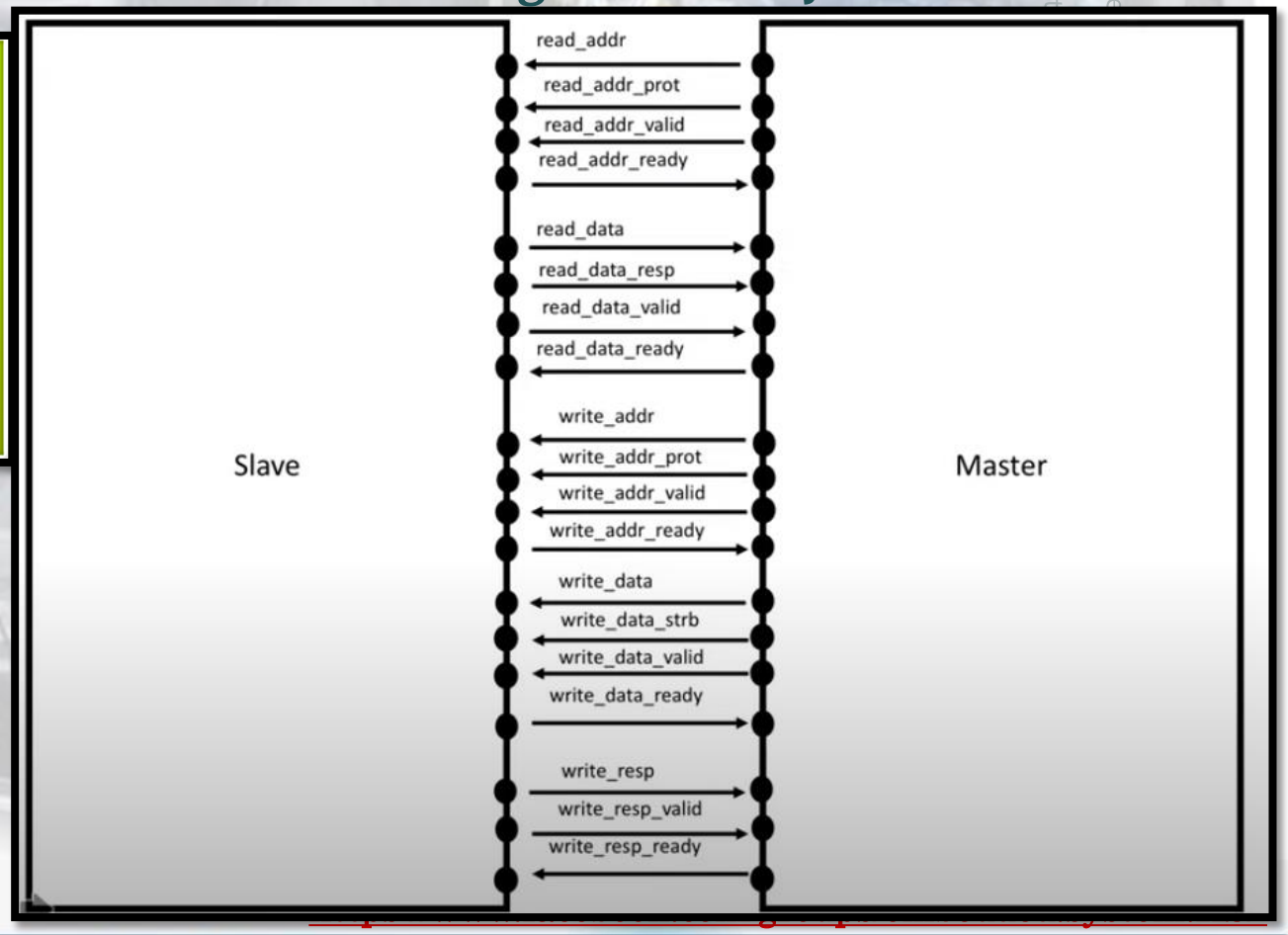
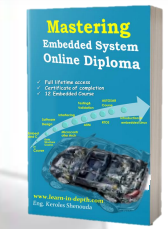
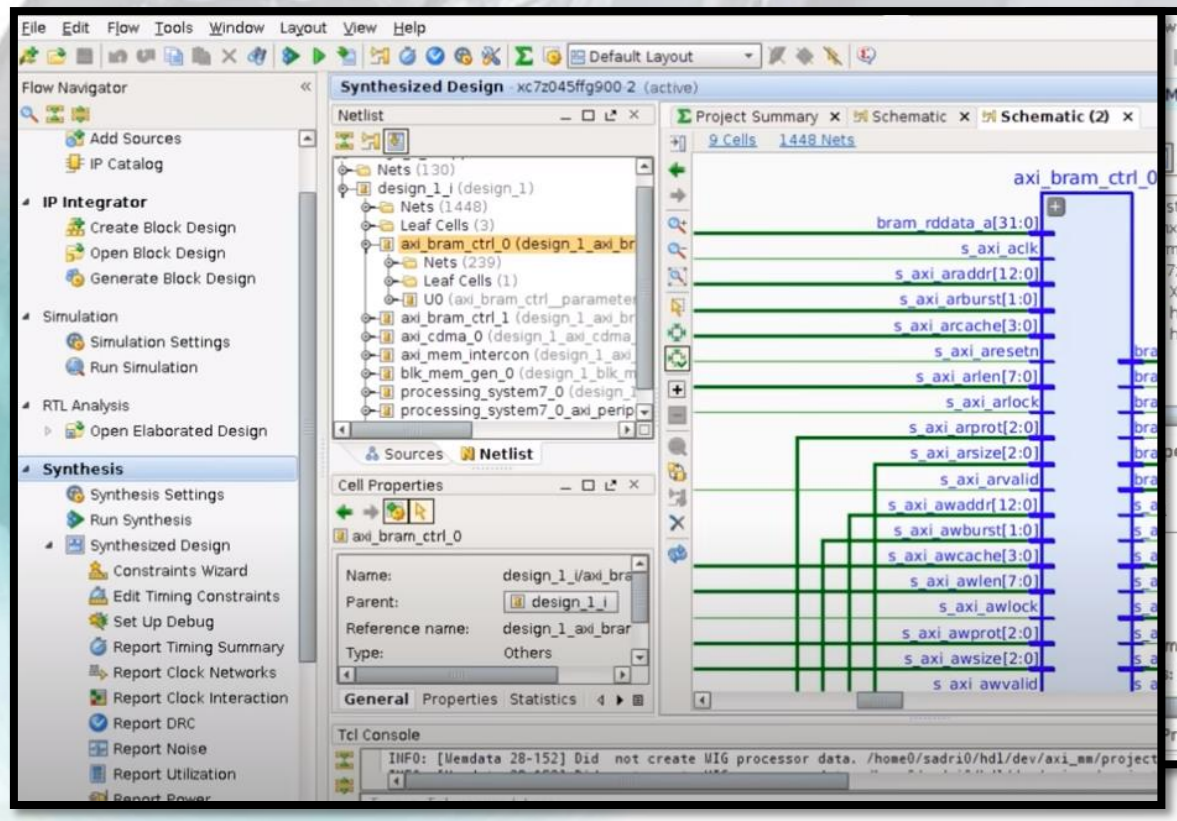


Table B1-1 AXI4-Lite interface signals

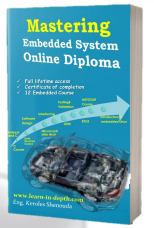
Global	Write address channel	Write data channel	Write response channel	Read address channel	Read data channel
ACLK	AWVALID	WVALID	BVALID	ARVALID	RVALID
ARESETn	AWREADY	WREADY	BREADY	ARREADY	RREADY
–	AWADDR	WDATA	BRESP	ARADDR	RDATA
–	AWPROT	WSTRB	–	ARPROT	RRESP



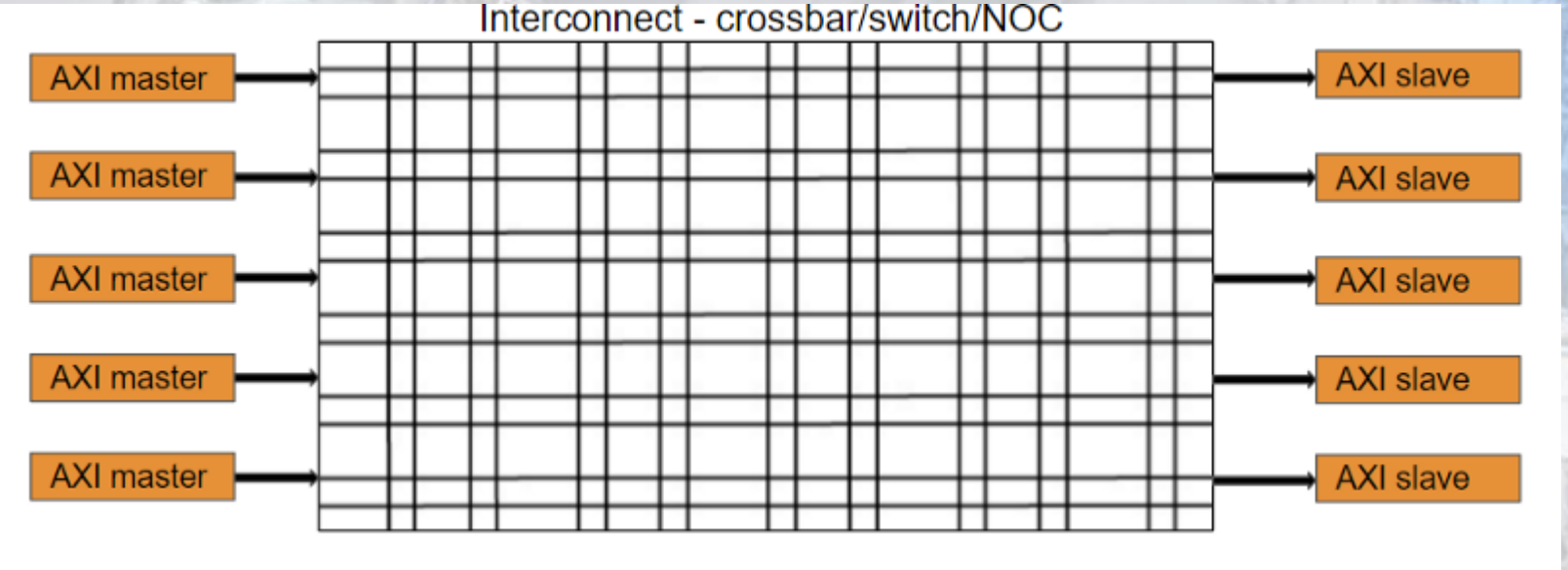
Real example
AXI Memory Mapped Interfaces & Hardware Debugging in Vivado

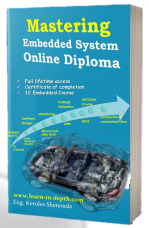


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



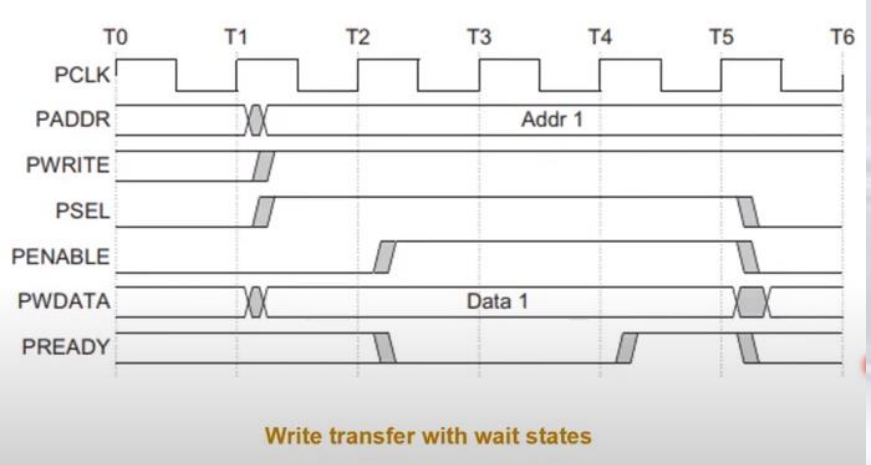
AXI





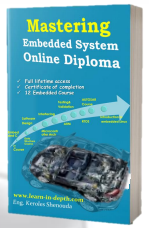
APB

- The Advanced Peripheral Bus (APB) is used for **connecting low bandwidth peripherals.**
- It is a simple non-pipelined protocol that can be used to communicate(read or write) from a bridge/master to a number of slaves through the shared bus



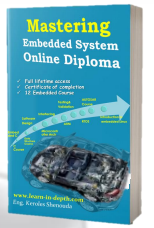
Write transfer with wait states

Signals	Source	Description
PCLK	Clock source	Clock
PRESETn	System bus equivalent	Reset. The APB reset signal is active LOW
PADDR	APB bridge	APB address bus, 32 bits wide
PPROT	APB bridge	Protection type. Indicates the normal/privileged, or secure/non-secure or data/instruction access.
PSELx	APB bridge	Slave select signal
PENABLE	APB bridge	Enable signal
PWRITE	APB bridge	Direction. When 1, write access else read access
PWDATA	APB bridge	Write data bus, 32-bits wide.
PSTRB	APB bridge	Write strobes. One write strobe bit for each 8-bits or 1-byte of the write data bus. Write strobes must not be active during read transfer.
PREADY	Slave interface	Ready signal, slave used this signal to extend an APB transfer
PRDATA	Slave interface	Read data, 32-bits wide
PSLVERR	Slave interface	Indicates transfer failure, slave error response.



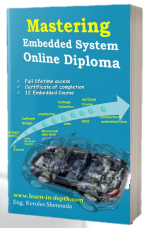
AHB

- The Advanced High-performance Bus (AHB) is used for connecting components that need **higher bandwidth on a shared bus**.
- These could be a internal memory or an external memory interface, DMA , DSP etc but the shared bus would limit the number of agents.
- Similar to APB, this is **a shared bus protocol** for multiple masters and slaves



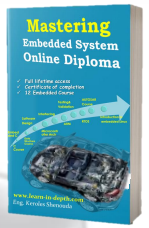
AHB-lite

- protocol is a simplified version of AHB.
- The simplification comes with support for only a single master design and that removes need for any arbitration, retry, split transactions etc.



AXI

- The Advanced Extensible interface (AXI) is useful for high bandwidth and low latency interconnects.
- The protocol also was an enhancement from AHB in terms of supporting multiple outstanding data transfers (pipe-lined), burst data transfers, separate read and write paths and supporting different bus widths.



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

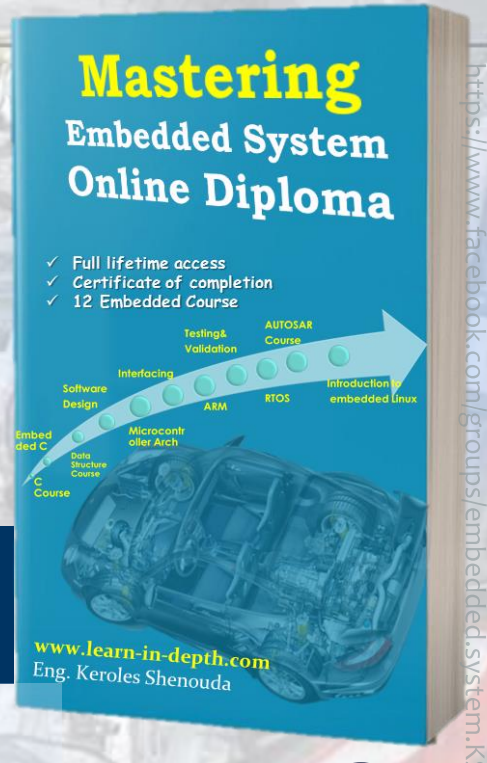
Apb, ahb, axi, … System BUSs

ADVANCED MICROCONTROLLER BUS ARCHITECTURE (AMBA)

ADVANCED PERIPHERAL BUS (APB)

ADVANCED EXTENSIBLE INTERFACE (AXI)

ADVANCED HIGH-PERFORMANCE BUS AHB

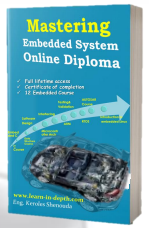


LEARN-IN-DEPTH

Be professional in embedded system

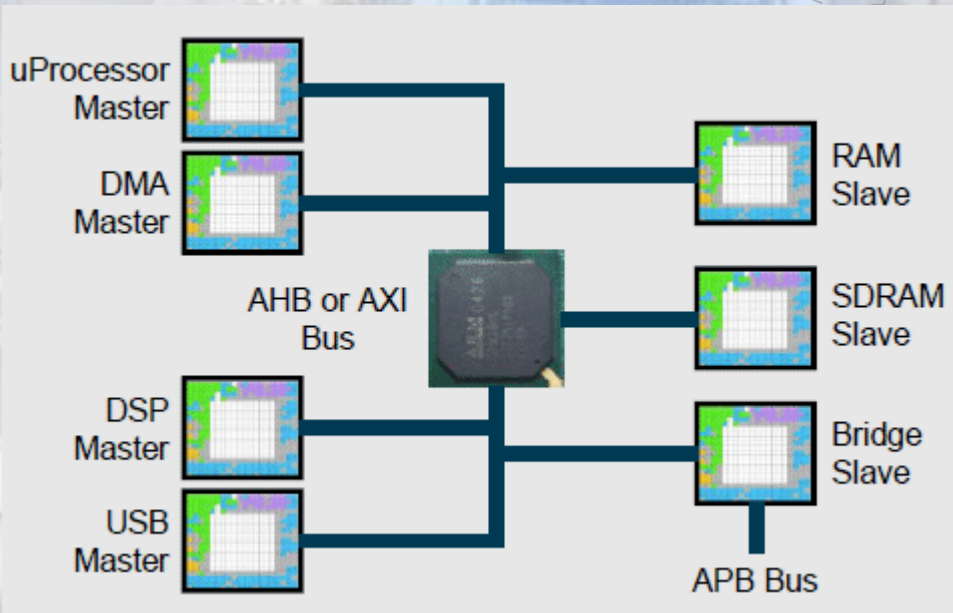
https://www.learn-in-depth.com/

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

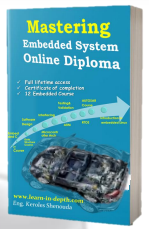


apb ahb axi

- For performance testing
  - The system model consists of the following:
  - uProcessor, DMA, DSP, and USB Master bus ports.
  - AHB or AXI Bus Arbiter.
  - RAM, SDRAM, and Bridge Slave bus ports.

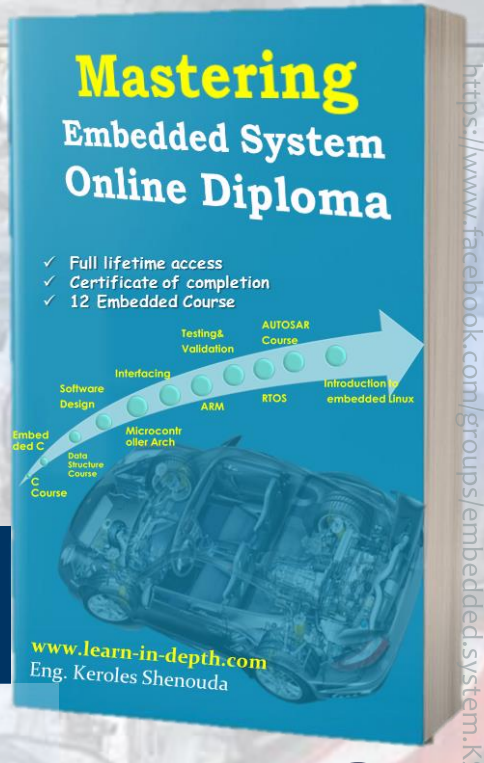


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

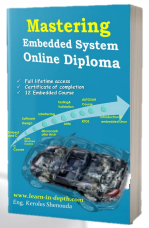
#Be_professional_in_embedded_system



Technical Questions about the BUS

LEARN-IN-DEPTH
Be professional in embedded system

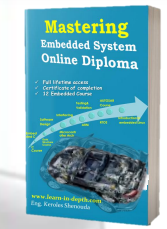
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Bus Questions

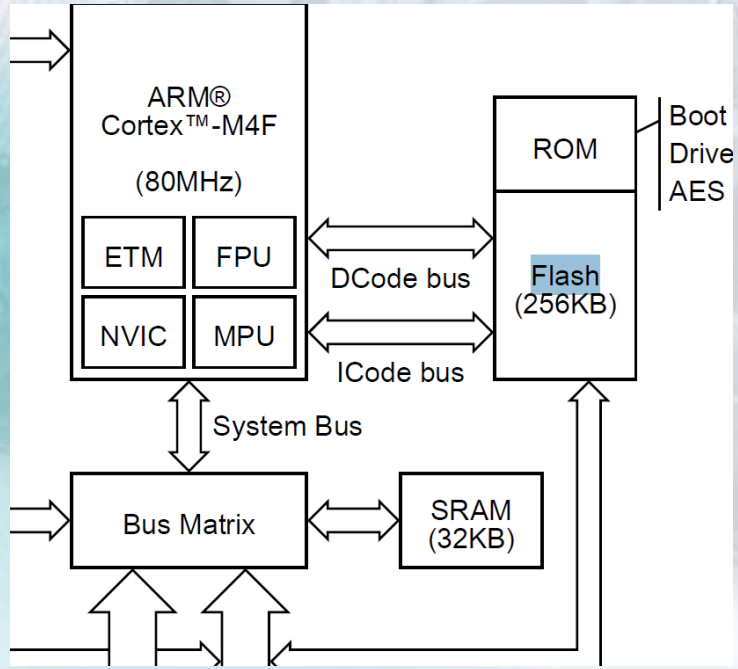
- In tm4c123
- Is it true that the system Bus is not connected with flash memory ?

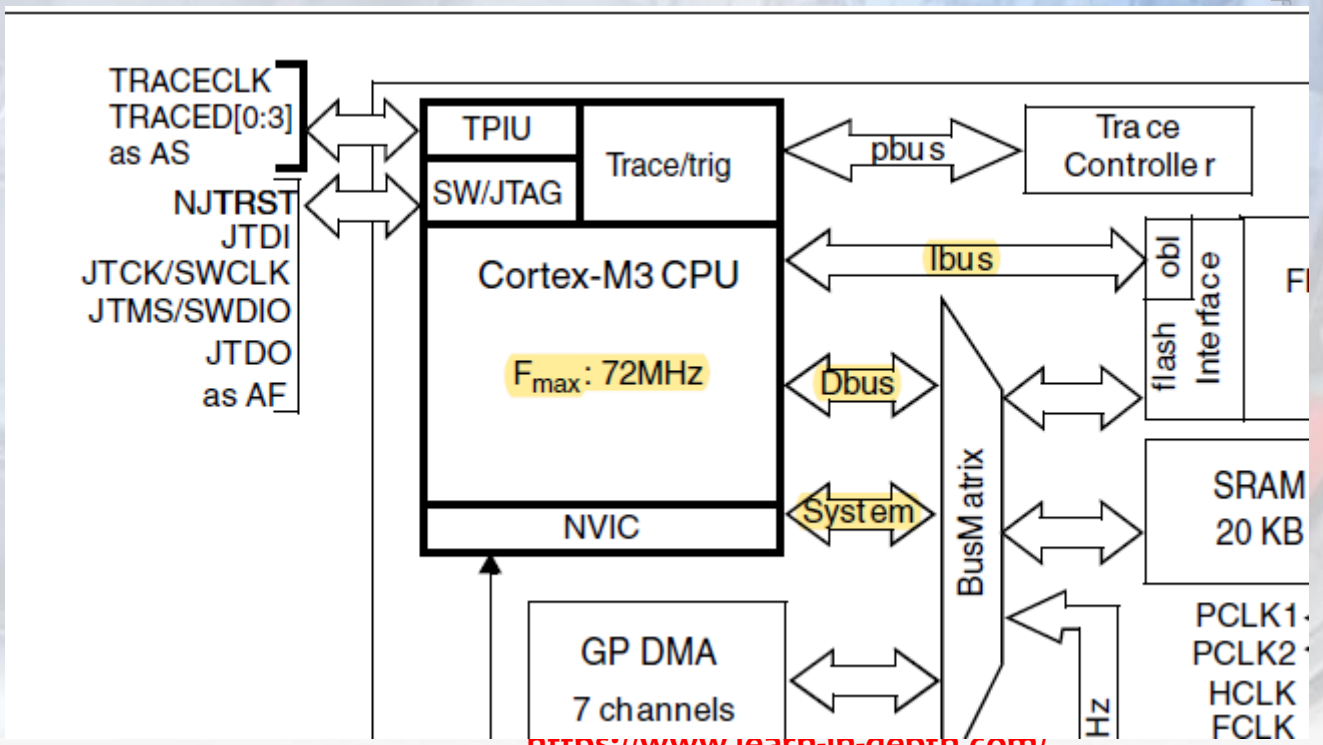
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

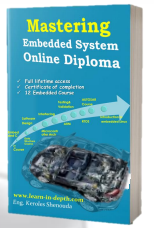


Bus Questions in stm32d103X/tm4c123

- Is it true that the system Bus is not connected with flash memory ?
- Yes this is true



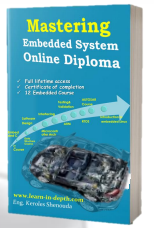




Bus Questions

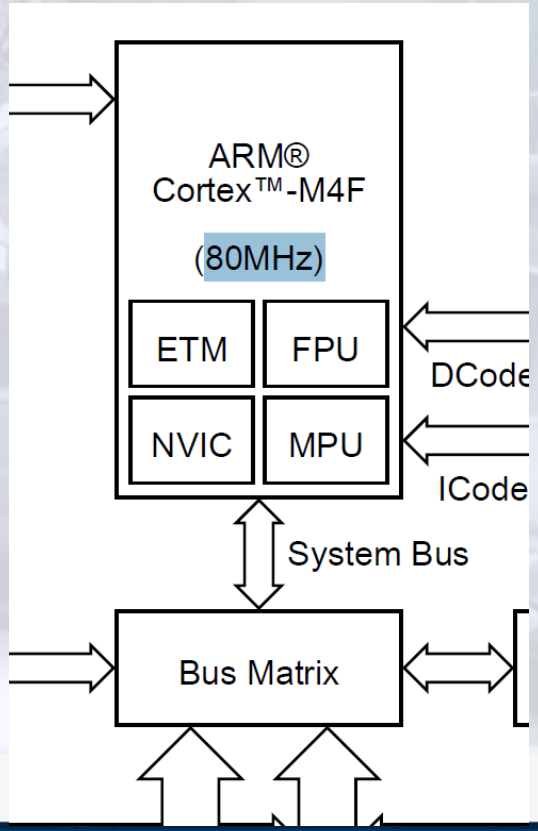
- In TM4C123 the system bus can operate at speed up to 80MHz T/F

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

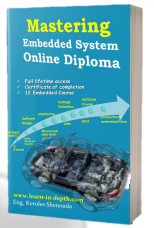


Bus Questions

- In TM4C123 the system bus can operate at speed up to 80MHz T/F
- Yes true



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



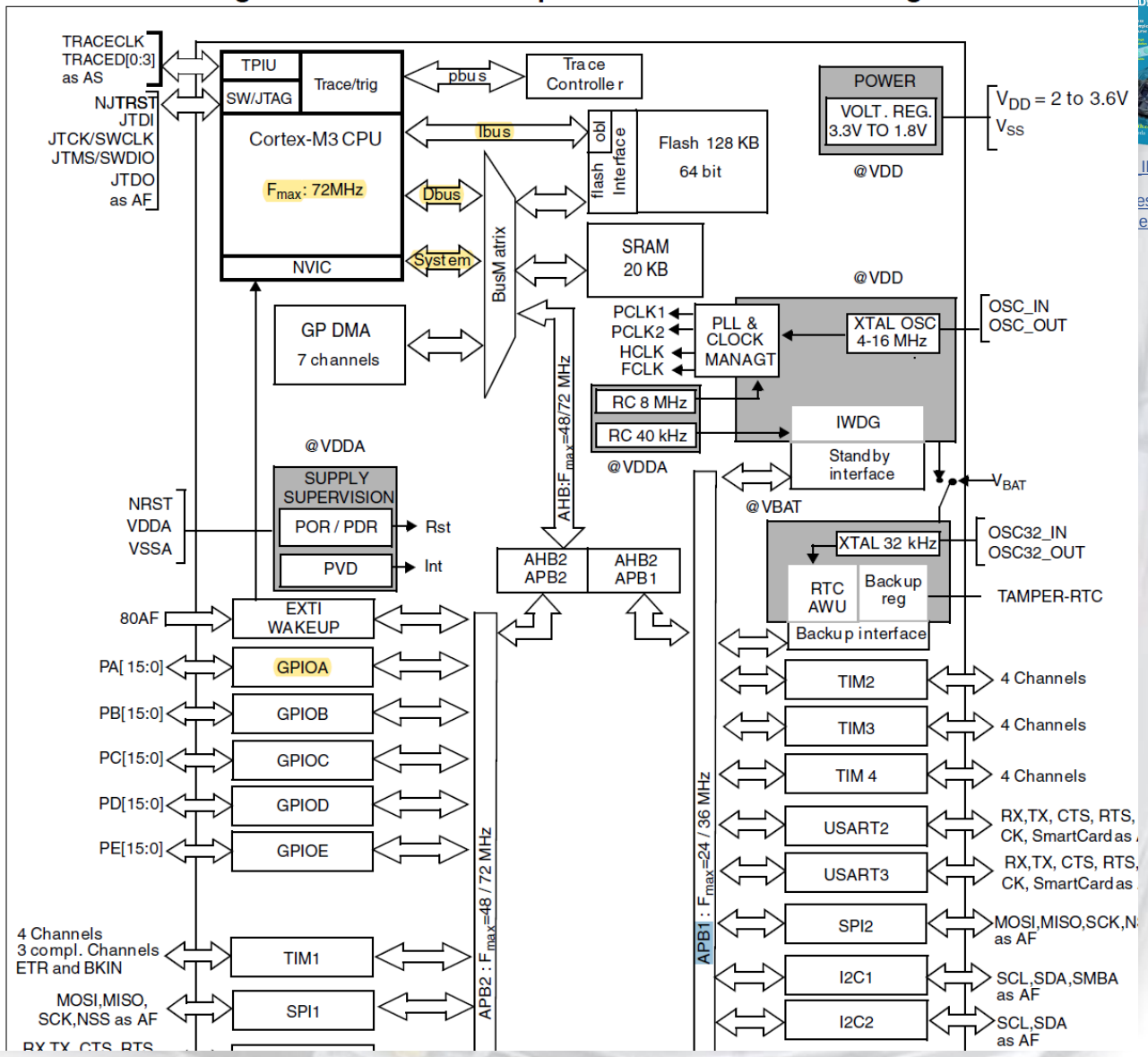
Bus Questions

- In stm32d103X, APB1 Bus can work with speed up to 72 MHZ T/F ?

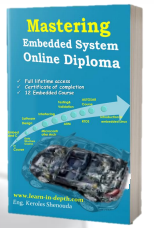
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Bus Questions

- In stm32d103X, APB1 Bus can work with speed up to 72 MHZ T/F ?
- False



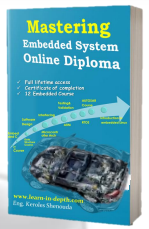
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Bus Questions

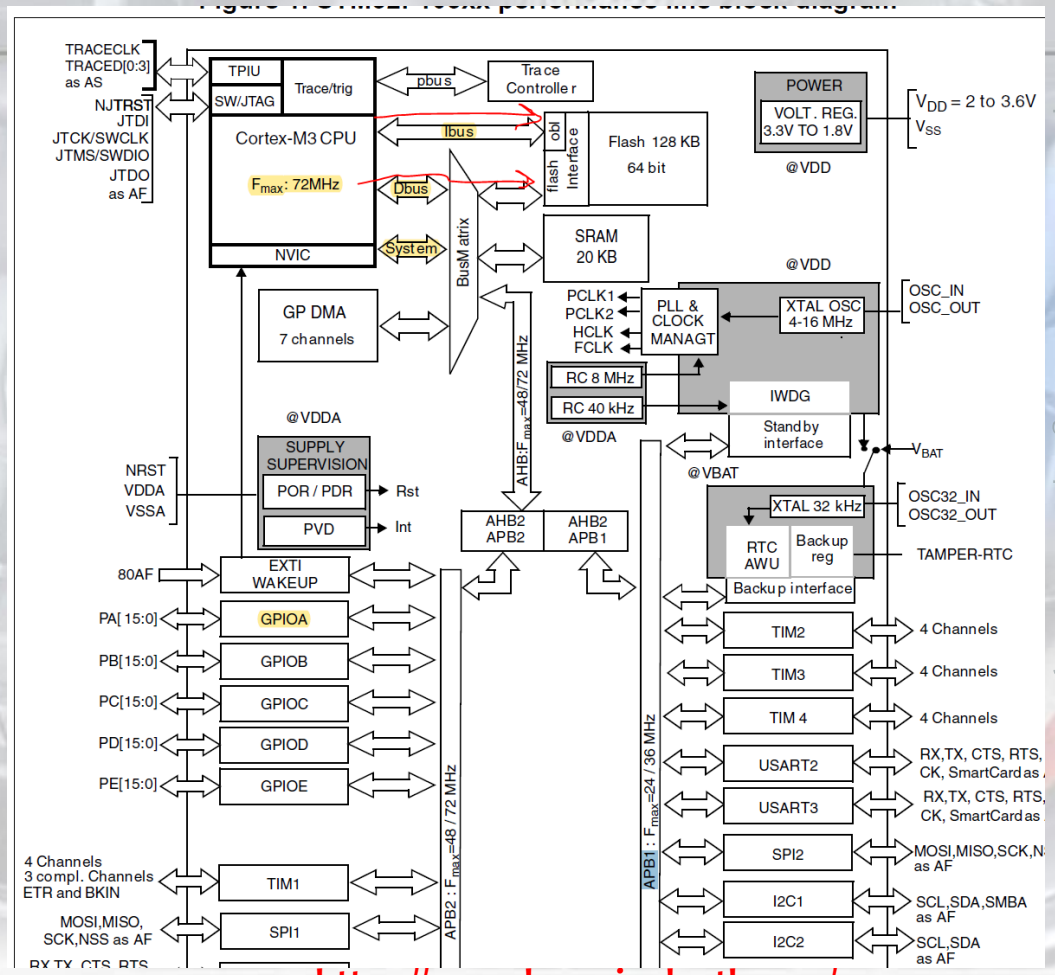
- In stm32d103X, CPU CAN fetch Instruction as well as data simultaneously from Flash Memory T/F ?

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

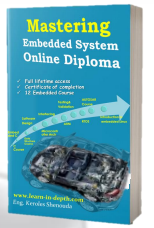


Bus Questions

- In stm32d103X, CPU CAN fetch Instruction as well as data simultaneously from Flash Memory T/F ?
- Yes true

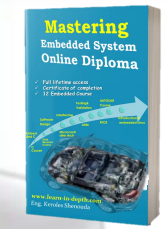


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



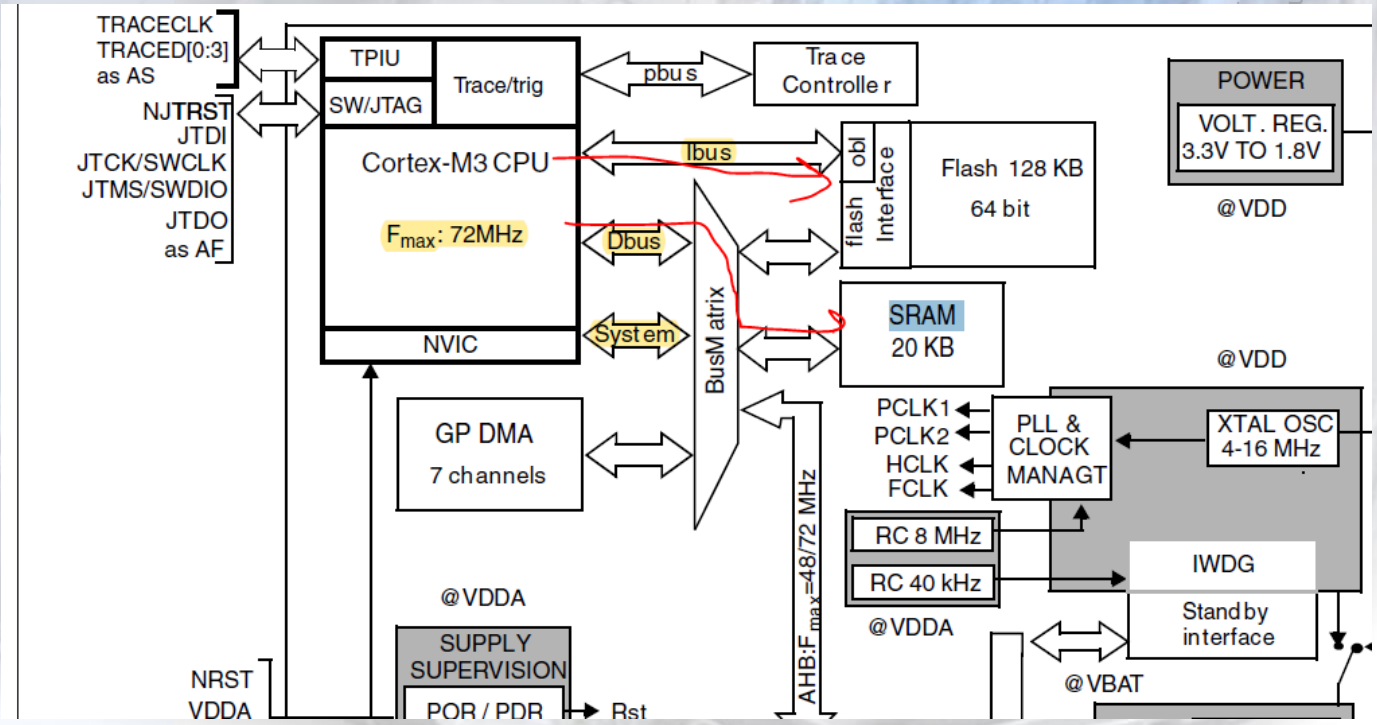
Question

- In stm32d103X, CPU CAN communicate with Flash as well as SRAM simultaneously T/F ?

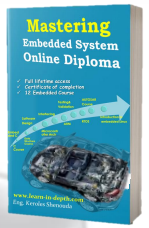


Question

- In stm32d103X, CPU CAN communicate with Flash as well as SRAM simultaneously T/F ?
- Yes true



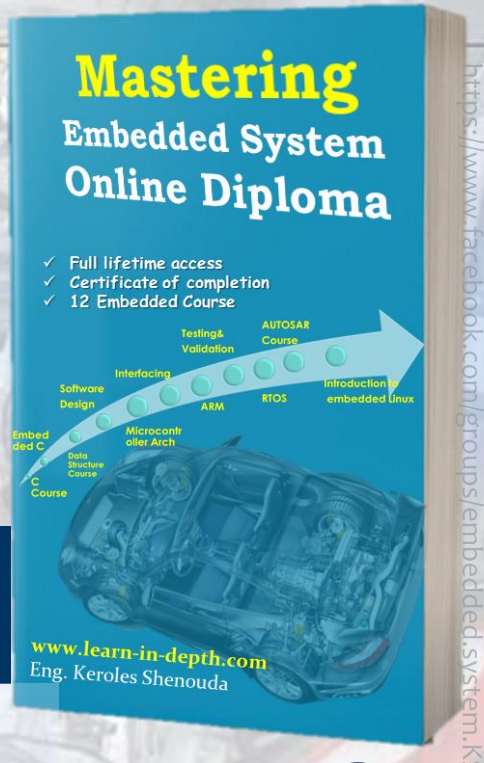
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

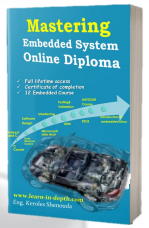
#Be_professional_in_embedded_system

BUS Matrix



LEARN-IN-DEPTH
Be professional in embedded system

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

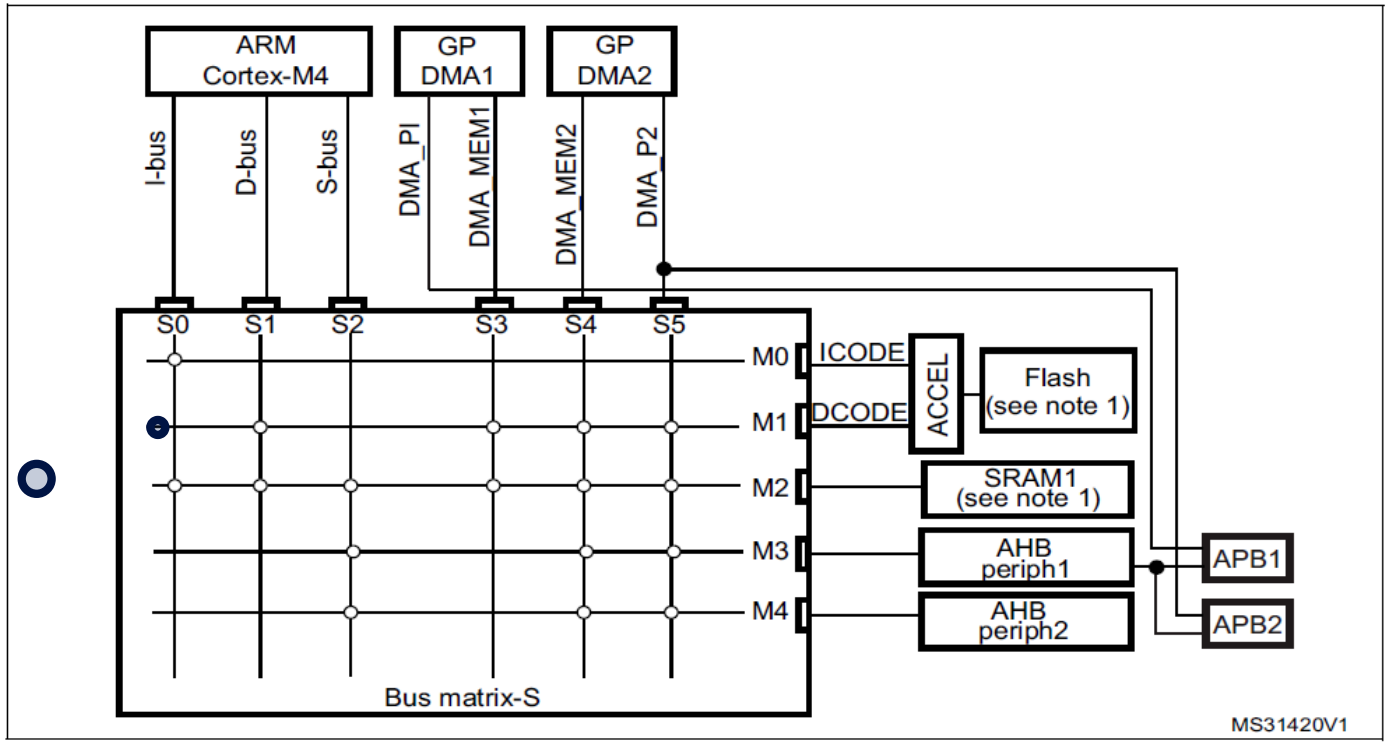


BUS Matrix

- With the help of the bus matrix, the main bus to the controlled bus can be accessed, so that even during the simultaneous operation of multiple high-speed peripherals, the system can achieve concurrent access and efficient operation

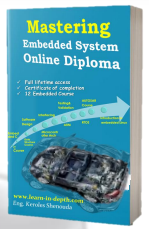
The I-Bus Slave port will not have access rule to M1

Figure 1. System architecture



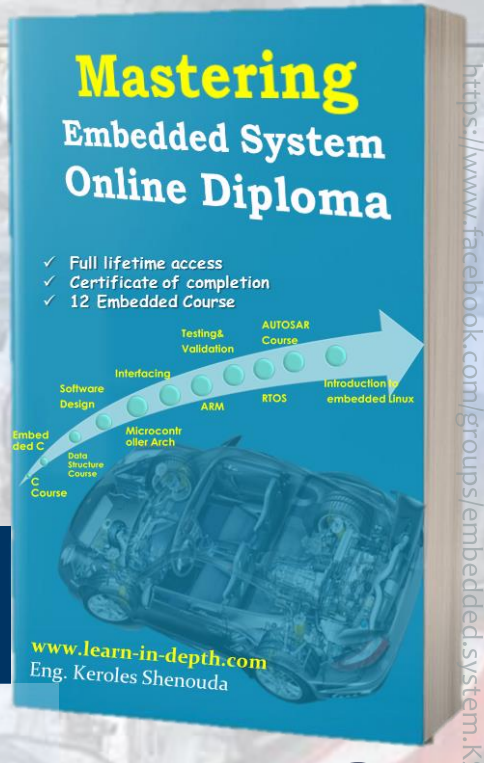
1. STM32F411xC/E: 256 KBytes / 512KBytes Flash with 128 KBytes SRAM

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



#LEARN_IN DEPTH

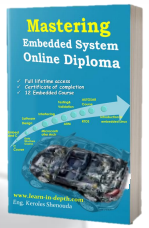
#Be_professional_in_embedded_system



Bit, Byte, Halfword, Word, Double-word and Nibbles

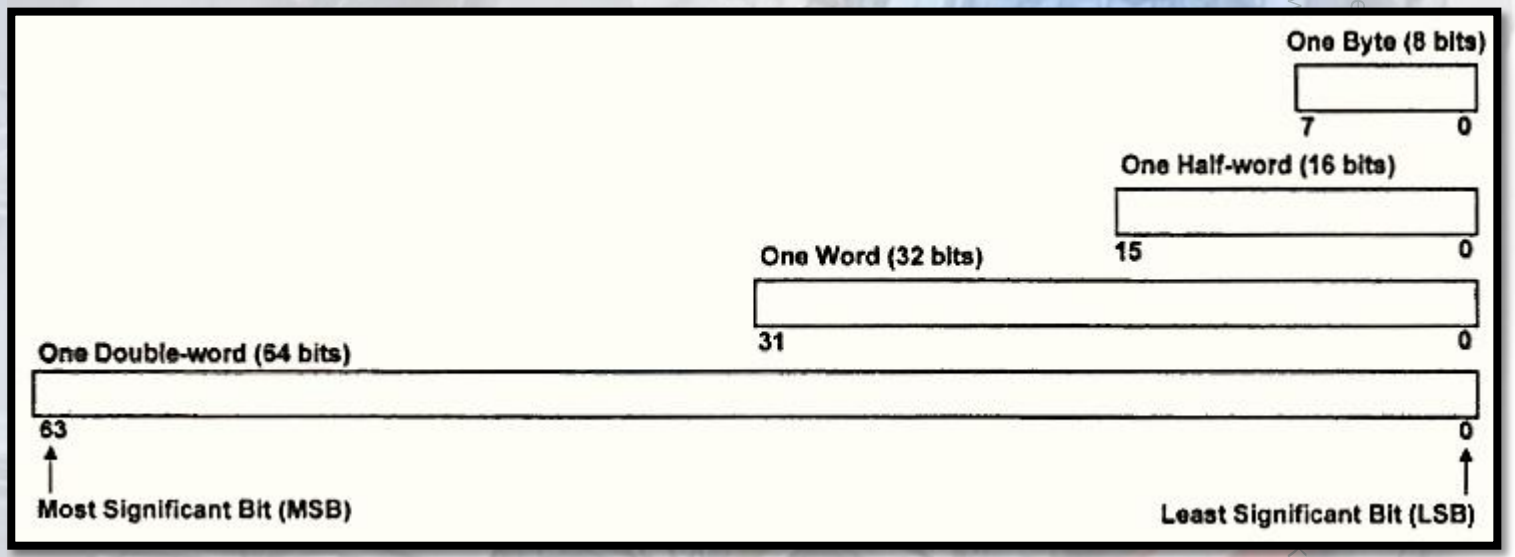
LEARN-IN-DEPTH
Be professional in embedded system

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

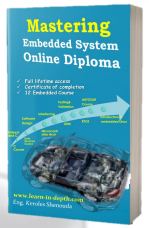


Bit, Byte, Halfword, Word, Double-word and Nibbles

- A Nibbles Is a group of 4bits
- a byte is a group of 8 bits,
- a *halfword* consists of 16 bits (or 2 bytes),
- a *word* has 32 bits (or 4 bytes), and
- a *double-word* contains 64 bits (or 8 bytes).
- **MSB and LSB may be located at some other positions**



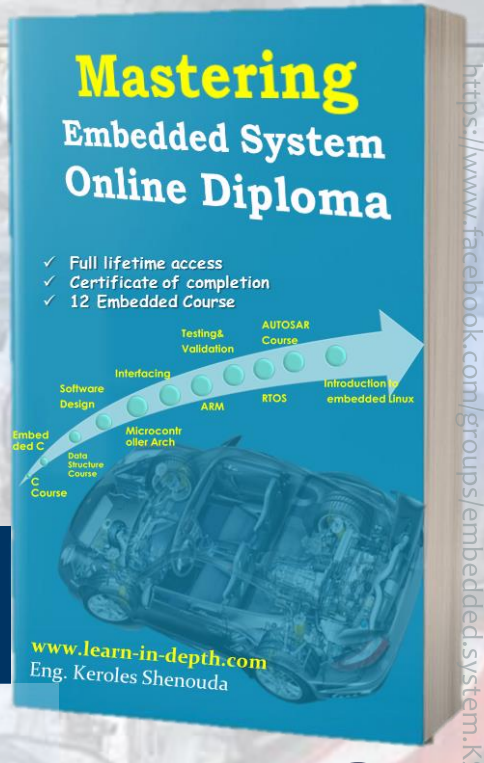
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

Big and Little Endian

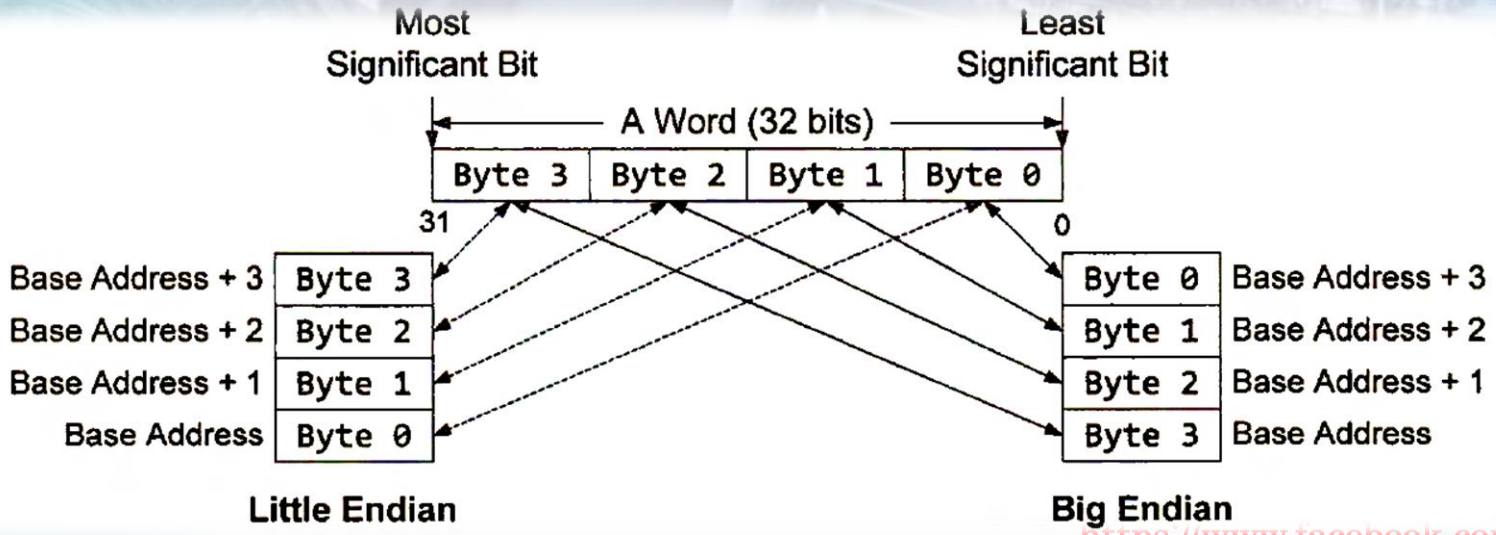


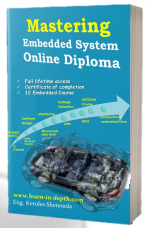
LEARN-IN-DEPTH
Be professional in embedded system

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

Big and Little Endian

- Little endian means the low-order byte of the number is stored in memory at the lowest address, and the high-order byte at the highest address.
  - **(The little end comes first.)**
- Big endian means the high-order byte of the number is stored at the lowest address, and the low-order byte at the highest address.
  - **(The big end comes first.)**

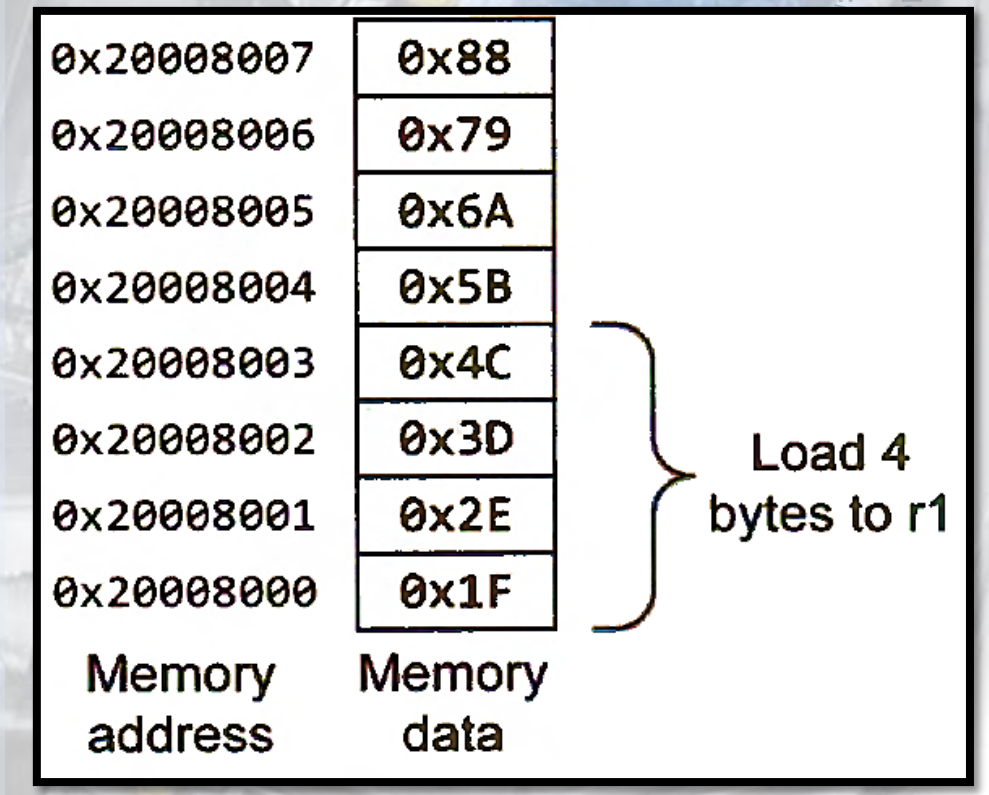




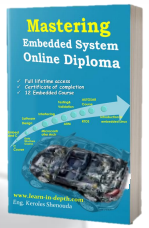
Big and Little Endian Example

- In this example, the assembly instruction
- "LDR rl, [r0]"
  - loads a 32-bit value from the memory address 0x20008000 to register rl.
- Register rl has different results, depending on whether the big or little endian is used.

- WHAT IS r1 value in case of
  - Little endian
  - big endian

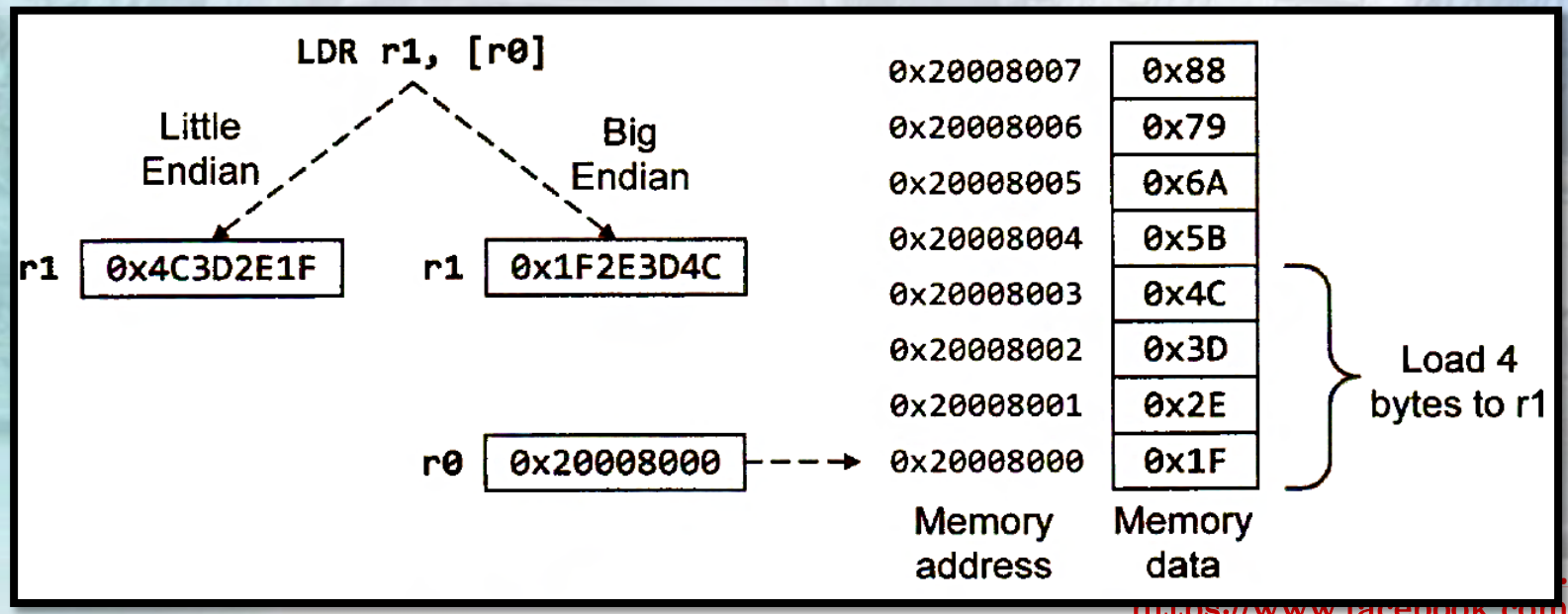


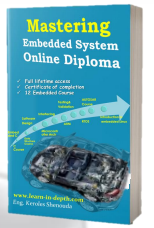
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Big and Little Endian Example

- In this example, the assembly instruction
- "LDR rl, [r0]"
  - loads a 32-bit value from the memory address 0x20008000 to register rl.
- Register rl has different results, depending on whether the big or little endian is used.





References

- https://docs.google.com/viewer?a=v&pid=sites&srcid=ZmtlLnV0bS5teXxyaWR6dWFuLXMtd2Vic2l0ZXxneDo2ODU0NzlkM2JkOTg4MjRk
- http://www.avrprojects.net/index.php/avr-projects/sensors/38-humidity-and-temperature-sensor-dht11?showall=&start=1
- http://www.cse.wustl.edu/~lu/cse467s/slides/dsp.pdf
- http://www.avr-tutorials.com/
- **Microprocessor: ATmega32 (SEE3223-10)**
  http://ridzuan.fke.utm.my/microprocessor-atmega32-see3223-10
- **http://circuitdigest.com/article/what-is-the-difference-between-microprocessor-and-microcontroller**
- **http://cs4hs.cs.pub.ro/wiki/roboticsisfun/chapter2/ch2_7_programming_a_microcontroller**
- Embedded Systems with ARM **Cortex-M Microcontrollers in** Assembly Language and C Dr. Yifeng Zhu Third edition June 2018

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


FOLLOW US
Press here
100
Mastering
Embedded System
Online Diploma
#LEARN_IN DEPTH
#Be_professional_in_embedded_system
Thank You
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/