FRA222 Microcontroller Interface

00 – INTRO ไงละ

Microcontroller Interface ???

Microcontroller

=CPU/Mircoprocesser

+Memory(RAM + ROM)

+Peripheral ?

+ฯลฯ

Interface

= shared boundary across which two or more separate components

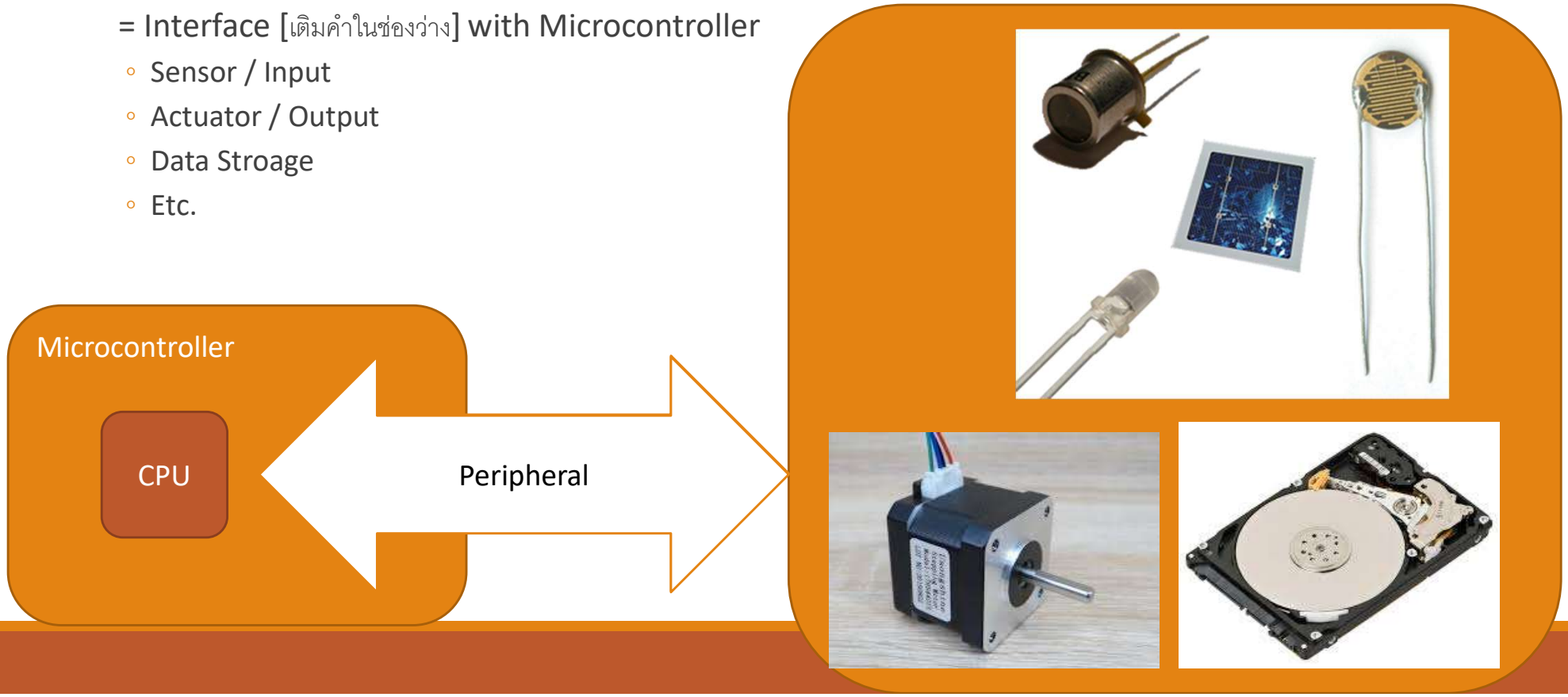
=ส่วนที่ใช้ติดต่อ/เชื่อมต่อ/แชร์กันระหว่างสิ่ง 2 สิ่ง

<< (Graphic) User Interface /Human Robot Interface

Microcontroller Interface

= Interface [เติมคำในช่องว่าง] with Microcontroller

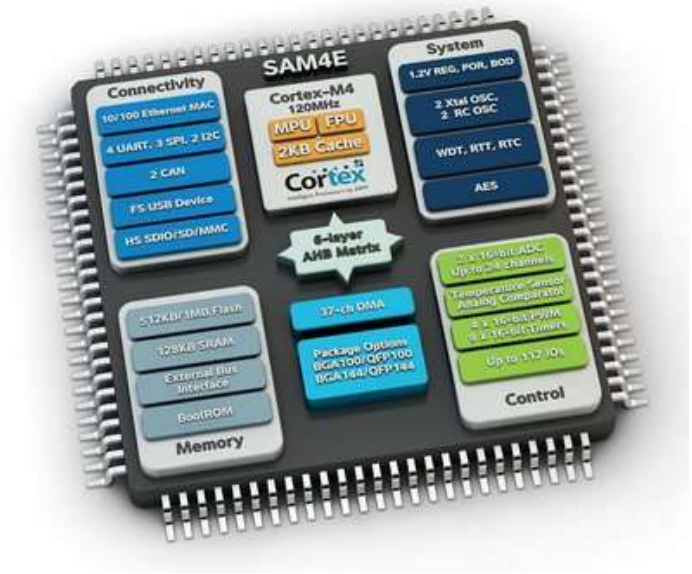
- Sensor / Input
- Actuator / Output
- Data Stroage
- Etc.



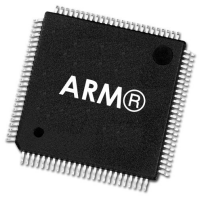
Peripheral

Auxiliary device used to put information into or/and get information out of and/or manage information in Microcontroller.

In this Class
GPIO
ADC
TIMER
UART
SPI
I2C
DMA
Etc.



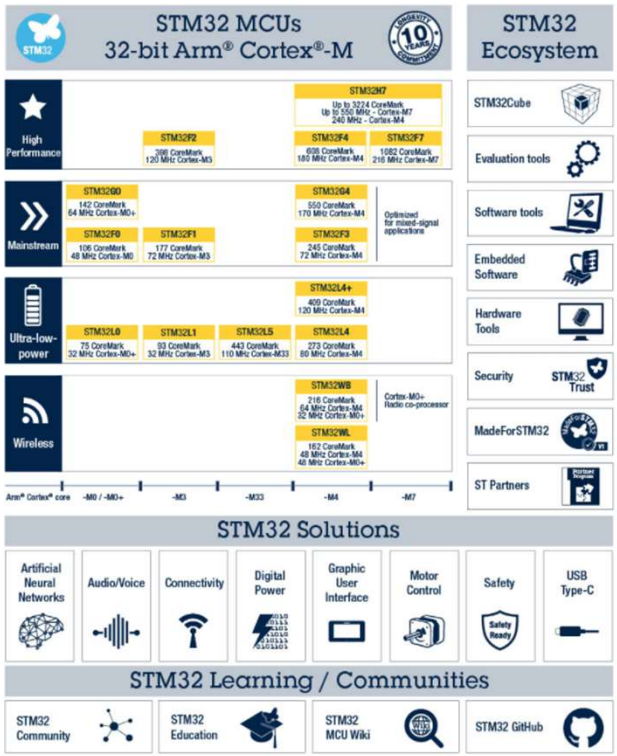
Processer family

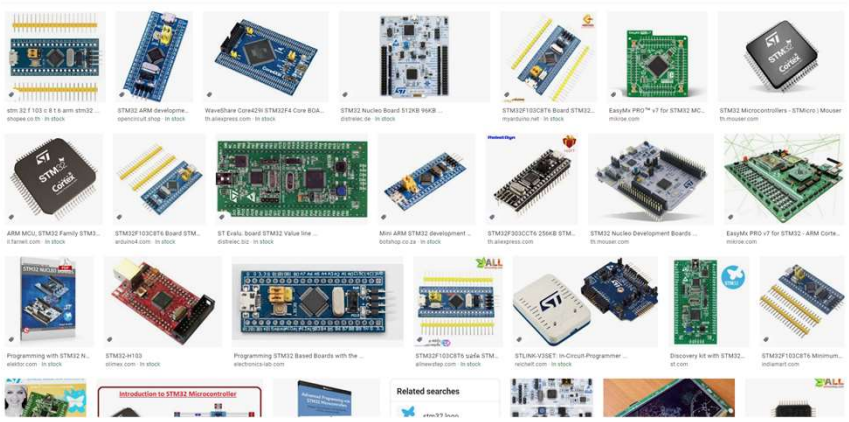


ARM (STM32)
AVR (ATMega/Arduino Uno)
PIC (PIC/dsPic)

4-bit	Am2900 · MARC4 · S1C6x · TLCS-47 · TMS1000 · µCOM-4
8-bit	6800 (68HC05 · 68HC08 · 68HC11 · S08 · RS08) · 6502 (65C134 · 65C265 · MELPS 740) · 78K · 8048 · 8051 (XC800) · AVR · COP8 · H8 · PIC10/12/16/17/18 · ST6/ST7 · STM8 · Z8 · Z80 (eZ80 · Rabbit 2000 · TLCS-870)
16-bit	68HC12/16 · C166 · CR16/C · H8S · MSP430 · PIC24/dsPIC · R8C · RL78 · TLCS-900 · Z8000
32-bit	Am29000 · ARM/Cortex-M (EFM32 · LPC · SAM · STM32 · XMC) · AVR32 · CRX · FR (FR-V) · H8SX · M32R · 68000 (ColdFire) · PIC32 · PowerPC (MPC5xx) · Propeller · SuperH · TLCS-900 · TriCore · V850 · RX · Z80000
64-bit	PowerPC64

STM32


STM32 MCUs
32-bit Arm® Cortex®-M
STM32 Ecosystem
High Performance
STM32H7
Up to 3224 CoreMark
Up to 550 MHz - Cortex-M7
240 MHz - Cortex-M4
STM32F2
398 CoreMark
120 MHz Cortex-M3
STM32F4
608 CoreMark
180 MHz Cortex-M4
STM32F7
1082 CoreMark
216 MHz Cortex-M7
Mainstream
STM32G0
142 CoreMark
64 MHz Cortex-M0+
STM32F0
106 CoreMark
48 MHz Cortex-M0
STM32F1
177 CoreMark
72 MHz Cortex-M3
STM32G4
550 CoreMark
170 MHz Cortex-M4
STM32F3
245 CoreMark
72 MHz Cortex-M4
Optimized for mixed-signal applications
Ultra-low-power
STM32L4+
409 CoreMark
120 MHz Cortex-M4
STM32L0
75 CoreMark
32 MHz Cortex-M0+
STM32L1
93 CoreMark
32 MHz Cortex-M3
STM32L5
443 CoreMark
110 MHz Cortex-M33
STM32L4
273 CoreMark
80 MHz Cortex-M4
Wireless
STM32WB
216 CoreMark
64 MHz Cortex-M4
32 MHz Cortex-M0+
Cortex-M0+ Radio co-processor
STM32WL
162 CoreMark
48 MHz Cortex-M4
48 MHz Cortex-M0+
Arm® Cortex® core
-M0 / -M0+
-M3
-M33
-M4
-M7
STM32Cube
Evaluation tools
Software tools
Embedded Software
Hardware Tools
Security
STM32 Trust
MadeForSTM32
ST Partners
STM32 Solutions
Artificial Neural Networks
Audio/Voice
Connectivity
Digital Power
Graphic User Interface
Motor Control
Safety
USB Type-C
STM32 Learning / Communities
STM32 Community
STM32 Education
STM32 MCU Wiki
STM32 GitHub


Related searches

STM32F411RE Resource + Limitation

Core: Arm® 32-bit Cortex®-M4 CPU with FPU frequency up to 100 MHz

Memories
- 512 Kbytes of Flash memory >> *(Program)
- 128 Kbytes of SRAM >> *(Variable)

A/D converter: up to 16 channels

8 timers
- 6 IC/OC/PWM or pulse counter and quadrature (incremental) encoder input

3 I2C

3 SPI

3 USART

See >> https://www.st.com/en/microcontrollers-microprocessors/stm32f411re.html

Tools (Software)

Main
- STM32CubeIDE
- Github(Optional)(ที่แปลว่าบังคับ)

Recommend
- Excel
- Microsoft Mathematics
- Calculator
- Matlab
- STM32CubeMonitor
- Logic2

GitHub - มีไว้เก็บงาน + ส่งงาน + backup งาน

Discord มีไว้ส่งงาน + ถามคำถาม **– ควรมีกล้องมือถือ หรือ คอมพิวเตอร์ที่มองเห็นได้ชัดๆ**

Classroom แหล่งรวมประกาศต่างๆ - โหลดแอปเอาไว้ก็ดี

In This Class (For now)

Wednesday/Thursday >> onsite Class ห้องคอมพิวเตอร์ FB306

- แนะนำให้ใช้โน้ตบุคส่วนตัวจะดีต่อใจกว่า

- ในกรณีไม่มีโน้ตบุค ให้ใช้ flash drive เซฟข้อมูลแทน *สร้างเป็น workspace ของตัวเอง

เนื้อหาเสริม (ถ้ามี) - ลงคลิปวันจันทร์

LAB (มีเป็นช่วงๆ) – ส่งใน 1 สัปดาห์ต่อไป [ทาง discord ช่วงกลางคืน / ส่งในคาบ]

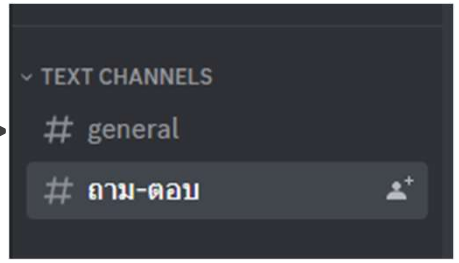
Exercise / Challenge – ไม่ต้องส่ง แต่ทำเพื่อฝึกทักษะ

Competency - ทดสอบความสามารถ

สงสัย?

ในคาบสั้นๆ – ถามเลย

ถ้ารู้สึกว่ายาว / สงสัยนอกคาบ >>



Now it time for Software

But first

- C
  - Variable and data container
  - Pointer
  - operator
- HAL

Variable and Data Container

Variable declaration

```
DataType VariableName = Value;
```

Data type and size

Character data type

- char(1B)

Integer data type

- int(4B)
  - Modifiler
    - signed
    - unsigned
    - short (2B)
    - long(4B)
    - long long(8B)

- integer type Aliases
  - integer type with width of exactly 8, 16, 32 and 64 bits respectively
    - int8_t / uint8_t
    - int16_t / uint16_t
    - int32_t/ uint32_t
    - int64_t/ uint64_t

Type specifier	Equivalent type	Bits
`short`	`short int` / `int16_t`	**16**
`short int`		
`signed short`		
`signed short int`		
`unsigned short`	`unsigned short int` / `uint16_t`	
`unsigned short int`		
`int`	`int` / `int32_t`	**32**
`signed`		
`signed int`		
`unsigned`	`unsigned int` / `uint32_t`	
`unsigned int`		
`long`	`long int` / `int32_t`	
`long int`		
`signed long`		
`signed long int`		
`unsigned long`	`unsigned long int` / `uint32_t`	
`unsigned long int`		
`long long`	`long long int` / `int64_t` (C++11)	**64**
`long long int`		
`signed long long`		
`signed long long int`		
`unsigned long long`	`unsigned long long int` /`uint64_t` (C++11)	
`unsigned long long int`		

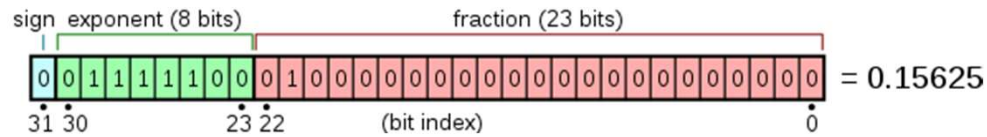
Type	Size in bits	Format	Value range	
			Approximate	Exact
character	8	signed		**-128** to **127**
		unsigned		**0** to **255**
integer	16	signed	**$\pm 3.27 \cdot 10^4$**	**-32768** to **32767**
		unsigned	**0** to **$6.55 \cdot 10^4$**	**0** to **65535**
	32	signed	**$\pm 2.14 \cdot 10^9$**	**-2,147,483,648** to **2,147,483,647**
		unsigned	**0** to **$4.29 \cdot 10^9$**	**0** to **4,294,967,295**
	64	signed	**$\pm 9.22 \cdot 10^{18}$**	**-9,223,372,036,854,775,808** to **9,223,372,036,854,775,807**
		unsigned	**0** to **$1.84 \cdot 10^{19}$**	**0** to **18,446,744,073,709,551,615**

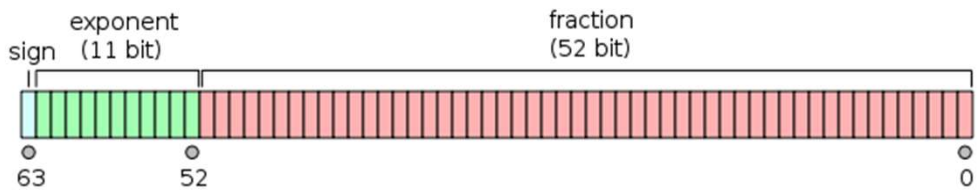
Data type and size

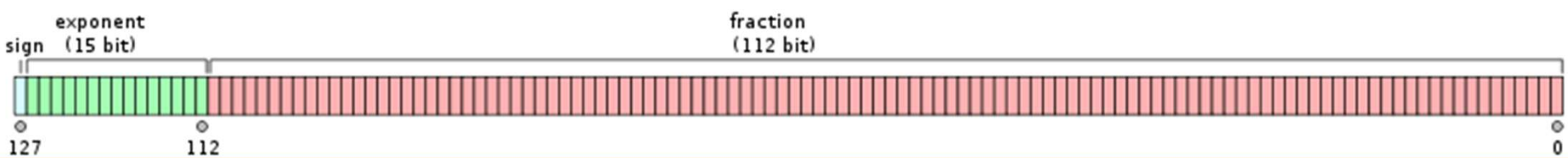
$$\text{value} = (-1)^{\text{sign}} \times 2^{(E-127)} \times \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i}\right)$$

floating-point data type

- float (32b) (~7.2 Digits)
- double (64b) (~15.9 Digits)
- long double (128b) (~34.0 Digits)







Type	Size in bits	Format	Value range	
			Approximate	Exact
binary floating point	32	IEEE-754	•min subnormal: **± 1.401,298,4 · 10^{-45}** •min normal: **± 1.175,494,3 · 10^{-38}** •max: **± 3.402,823,4 · 10^{38}**	•min subnormal: **±0x1p-149** •min normal: **±0x1p-126** •max: **±0x1.fffffep+127**
	64	IEEE-754	•min subnormal: **± 4.940,656,458,412 · 10^{-324}** •min normal: **± 2.225,073,858,507,201,4 · 10^{-308}** •max: **± 1.797,693,134,862,315,7 · 10^{308}**	•min subnormal: **±0x1p-1074** •min normal: **±0x1p-1022** •max: **±0x1.fffffffffffffp+1023**
	80	x86	•min subnormal: **± 3.645,199,531,882,474,602,528 · 10^{-4951}** •min normal: **± 3.362,103,143,112,093,506,263 · 10^{-4932}** •max: **± 1.189,731,495,357,231,765,021 · 10^{4932}**	•min subnormal: **±0x1p-16446** •min normal: **±0x1p-16382** •max: **±0x1.ffffffffffffffffep+16383**
	128	IEEE-754	•min subnormal: **± 6.475,175,119,438,025,110,924, 438,958,227,646,552,5 · 10^{-4966}** •min normal: **± 3.362,103,143,112,093,506,262, 677,817,321,752,602,6 · 10^{-4932}** •max: **± 1.189,731,495,357,231,765,085, 759,326,628,007,016,2 · 10^{4932}**	•min subnormal: **±0x1p-16494** •min normal: **±0x1p-16382** •max: **±0x1.ffffffffffffffffffffffffffff p+16383**

Data type and size

Pointer data type

Store Address

- int*
- float*
- void*
- uint32_t* ptr;

- Operator
  - Reference &a
    - get **memory address** form **variable**
    - uint8_t* ptr = &a ;
  - Dereference *a
    - get/set **value** form **pointer**
    - *ptr = 300 ;

This is RAM (Very Minimal model)

Address	0	1	2	3	4
A					
B					
C					
D					

This is RAM (Very Minimal)

```
int i = 10;
```

Address	0		2	3	4
A	i 10				
B					
C					
D					

This is RAM (Very Minimal

```
int i = 10;
int j = 20;
```

Address	0	1	2	3	4
A	i 10	j 20			
B					
C					
D					

This is RAM (Very Minimal

```
int i = 10;
int j = 20;

int *x = &i
```

int *x (ประกาศตัวแปรชนิดpointer)

&[Variable] เป็นการขอaddress ของตัวแปรนั้นๆ

Address	0				4
A	i 10	j 20			
B					
C					
D					

This is RAM (Very Minimal

```
int i = 10;
int j = 20;

int *x = &i;
```

Address	0	1	2	3	4
A	i 10	j 20	x A0		
B					
C					
D					

This is RAM (Very Minimal

```
int i = 10;
int j = 20;

int *x = &i
print(x);
```

Address	0	1	2	3	4
A	I 10	j 20	X A0		
B					
C					
D					

- x = A0

This is RAM (Very Minimal

```
int i = 10;
int j = 20;

int *x = &i
print(x);
print(*x);
```

Address	0	1	2	3	4
A	i 10	j 20	x A0		
B					
C					
D					

- x = A0

This is RAM (Very Minimal

```
int i = 10;
int j = 20;

int *x = &i
print(x);
print(*x);
```

Address	0	1	2	3	4
A	i 10	j 20	x A0		
B					
C					
D					

- x = A0
- *x = 10

This is RAM (Very Minimal mode

```
int i = 10;
int j = 20;

int *x = &i
print(x);
print(*x);

*x =30;
print(i);
```

Address	0	1	2	3	
A	i 10	j 20	x A0		
B					
C					
D					

- x= A0
- *x = 10

This is RAM (Very Minimal mode

```
int i = 10;
int j = 20;

int *x = &i
print(x);
print(*x);

*x =30;
print(i);
```

Address	0	1	2	3	
A	i 30	j 20	x A0		
B					
C					
D					

- x = A0
- *x = 10
- i = 30

Pointer

ประกาศตัวแปร นำหน้า ด้วย *

- int *a;
- float *b;
- uint32_t *c;

หลังจากประกาศแล้ว ใส่ *คือค่าของ ตัวแปรในตำแหน่งที่ pointer เก็บไว้ แต่ถ้าไม่ใส่ จะได้ address ที่ pointer เก็บไว้

- เช่นจากตัวอย่าง *x จะได้ค่า ของ i ซึ่งก็คือ 30
- แต่ x จะได้ ตำแหน่งของ i ซึ่งก็คือ A0

&แล้วตามด้วย variable (ไม่มีเว้นวรรคนะ ถ้ามีจะเป็น Bitwise And แทน) จะได้address ของ ตัวแปรนั้นๆ

ในความเป็นจริง ของstm32 pointer มีขนาด 32 bits เพราะฉะนั้น x ใน stm32 จริงๆ จะอารมณ์ประมาณว่า... x = 0x2B3C14FF;

Data type and size

data type **void**

- no type ➔ Pointer with no specific type
  - void* ptr;
- no value ➔ function with no return value
  - void toggleLed()
- no parameter ➔ function with no parameter
  - int GetDate(void)

Array

declare ➔ DataType ArrayName [ArraySize] = {data};

access ➔ ArrayName [Position] , *(ArrayName+Position)

ArrayName ➔ Pointer to Array

&ArrayName[0] / &ArrayName ➔ Pointer to first element of Array

sizeof(ArrayName) ➔ size of whole array

sizeof(&ArrayName[0]) ➔ size of array element

enum– Enumeration – assign name integral constant

```
enum EnumTag
{
        member1,
        member2,
        member3,….
        …,
        memberN
} enumVar;
```

```
enum DaysOfWeek
{
    Sun,
    Mon,
    Tue,
    Wed,
    Thu,
    Fri,
    Sat
}Day;
```

enum – Enumeration
– assign name integral constant

```
enum DaysOfWeek
{
        Sun,Mon,Tue,Wed,Thu,Fri,Sat
};

enum DaysOfWeek Day;

Day = Mon;

int WorkHours[7];
WorkHours[Mon] = 8;
```

Sun = 0,Mon =1,Tue =2,…

declare variable with enum ,act like declare int

Day = 1

enum

```
enum DaysOfWeek
{
        Sun=5,Mon,
        Tue,Wed,
        Thu=20,Fri,Sat
};
```

Sun = 5,Mon =6,Tue =7,Wed =8,
Thu =20,Fri = 21,Sat =22

```
Day = Mon;
```

Day = 6

enum

```
enum StateOfMachine
        {
                IDLE,
                INIT,
                WORKING,
                CLEANING,
                FAIL
        }
```

```
enum {LO ,HI};

enum{LOW,MID=128,HI=255};
```

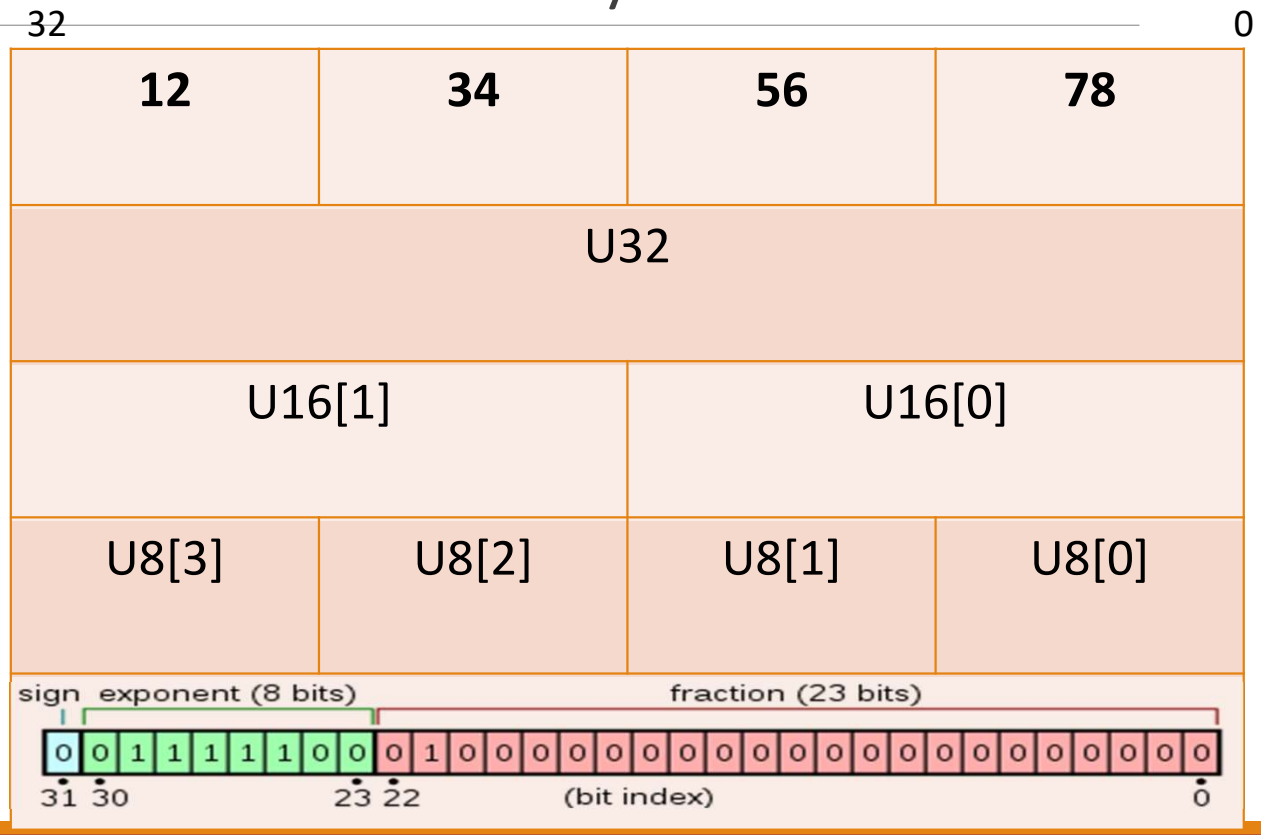
Union
-Share datatype in same memory location

```
union UnionTag
{
        Datatype Member1;
        Datatype Member2;
        Datatype Member3;
        …
         Datatype MemberN;
} UnionVar;
```

```
union U32BitsConv
{
        uint32_t U32;
        uint16_t U16[2];
        uint8_t U8[4];
        float   F32;
} U32Convert;
```

Union
-Share datatype in same memory location

```
union U32BitsConv
{
        uint32_t U32;
        uint16_t U16[2];
        uint8_t U8[4];
        float    F32;
} U32Convert;

U32Convert.U32 = 0x12345678
```



Structure - declare

```
struct StructTag
{
        int a;
        float b;
        char c[3];
};

struct StructTag structName = { 123 , 3.14 , {'P', 'u', 'n'} } ;
struct StructTag struct2;

struct2 = structName ; //Copy struct
```

```
struct StructTag{
        int a;
        float b;
        char c[3];
} structName = { 123 , 3.14 , {'P', 'u', 'n'} } ;
```

Structure access

```
struct StructTag{
        int a;
        float b;
        char c[3];
} structName = { 123 , 3.14 , {'P', 'u', 'n'} } ;

struct StructTag* ptrToStruct = &structName
```

```
structName.a = 456;
float EstPi = structName.b;
structName.c[1] = 2

ptrToStruct->a =456;
(*ptrToStruct).a =456;
```

typedef - give Type a new Name

➔ typedef typeName NewTypeName;

```
typedef uint8_t Byte;
Byte Var = 255;

typedef unsigned char uint8_t;

typedef uint8_t MachineState;
MachineState State=0;
```

```
struct Date
{
        unsigned int day ;
        unsigned int mouth ;
        unsigned int year ;
} ;

struct Date stDate = {31,12,2021} ;
```

```
typedef struct Date
{
        unsigned int day ;
        unsigned int mouth ;
        unsigned int year ;
}DateStructure ;

DateStructure stDate = {31,12,2021} ;
```

```
union U32BitsConv
{
	uint32_t U32;
	uint16_t U16[2];
	uint8_t U8[4];
	float	F32;
} ;

union U32BitsConv U32convert;
```

```
enum DaysOfWeek
{
	Sun,Mon,Tue,Wed,Thu,Fri,Sat
};

enum DaysOfWeek Day;
```

```
typedef union U32BitsConv
{
	uint32_t U32;
	uint16_t U16[2];
	uint8_t U8[4];
	float	F32;
} U32Convertor;

U32Convertor U32convert;
```

```
typedef enum __DaysOfWeek
{
	Sun,Mon,Tue,Wed,Thu,Fri,Sat
}DayOfWeek;

DaysOfWeek Day;
```

Variable Storage Class

	Scope	Life
auto	within block	End of block
static	within block	End of program
extern	Multiple file / Global	End of program
register	within block	End of block

auto – Default Storage Class ,and Automatically assign to any variable(that not use static extern or register)

static - preserving value until end of program

extern – This Variable define somewhere else but not this file

Register – If you(compiler) can , assign this variable to CPU Register

static

```
int countUp()
{
        static int a;
        return a++;
}
```

by default , static variable Initial value is 0

A preserved value when end of function

extern

Tell complier that b is declare in another file

```
extern int b;

void countUp()

{

        extern int a;

        return a++;

}
```

Tell complier that a is declare in another file , but only visible by this function

Main.c

```
TIM_HandleTypeDef htim1;
UART_HandleTypeDef huart2;
```

User.cpp

```
extern TIM_HandleTypeDef
htim1;
extern UART_HandleTypeDef
huart2;
```

Register

```
register int i ;
for(i=0;i<10;i++)
{
    //do something
}
```

Compiler will put i in CPU register instead of RAM (If they can)

Register variable can’t access by pointer

Register can’t used in global scope

Used when variable need frequently access and used in small size

Variable type qualifiers – Tell Complier something important (for better optimize)

const – DO NOT EDIT THIS

volatile – This can change form other things (like hardware)

const – DO NOT EDIT THIS

Tell compiler that “This variable value will not be change” , make variable can store in Read-only Memory (sometime inside code directly)

const float Pi = 3.1415;

const int *ptr = &i
ptr is pointer to constant
“*ptr” **can’t** modified , “ptr” can modified
“i” can or cannot be constant

int const *ptr = &i
ptr is pointer to constant int
“*ptr” **can’t** modified , “ptr” can modified
“i” must be constant int

int *const ptr = &i
ptr is constant pointer to int
“*ptr” can modified , “ptr” **can’t** modified
“i” can or cannot be constant

const int *const ptr = &i
ptr is constant pointer to constant int
“*ptr” **can’t** modifild , “*ptr” **can’t** modified
“i” must be constant int

volatile – This can change form other things

Tell compiler that “this variable can be edit by someone that not in this program at anytime”

➔ compiler not optimize this variable out and try to read every time we call it.

Variable Initialization and constants

Variable should Initial before use to avoid garbage data
(expect static and global that set to 0 by compiler)

```
int a = 1;
int b[]= {1,2,3}
char c = 'g';
float f = 1.0;
char h[] ="Hello World"
int g[10] ={0}
```

' ' convert ASCII to Number

.0 make floating point constant
useful in calculation
1 / 3 = 0 Because int / int = int
but 1/3.0 = 0.333333 because int / float = float

" " make array of char constant
size is 12 because of white space and Null

initial all element set to 0

Variable Initialization and constants

```
uint8_t a = 255U;          //Unsigned
uint16_t b = 254u;         //Unsigned
uint8_t c = 0xFF;          //Hex
uint8_t d = 056;           //Oct
uint8_t e = 0b01001100;    //Bin
int32_t f = -123L;         //Long
int32_t g = 123ul;         //Unsigned long
uint64_t g= -1LL;          //Long Long
float h = 2.99792458E8 ;   // 2.99792458x10^8
```

Variable Initialization and constants

```
struct StructTag{
        int a;
        float b;
        char c[3];
} structName = { 123 , 3.14 , {'P', 'u', 'n'} } ;
```

```
struct StructTag{
        int a;
        float b;
        char c[3];
} structName = { 0} ;
```

Variable Name

We should Name Variable properly to show other people (and yourself in next week)What this variable does.

We have auto complete, don’t scare to name it a little bit longer

Common rule
-contain letters ,digits , Underscore
-begin with letter or underscore
-no whitespace or special character
-not reserved word
-Case sensitive

some common style and tip

Macro and Constant value ➔ ALL_UPPER_CASE_WITH_UNDERSCORE

something that shouldn't access by anyone ➔ _something

function and variable name ➔ wordGroup_wordGroup

pointer ➔ ptrVariable or p_Variable

some quick loop ➔ i,j,k are fine ^ ^

tips: add some common name at start of variable to easy auto correct

- like HAL_ADC_XXXX

Preprecessor – Do before compile (marco)

#include ➔ include library can be .h ,.c , .hpp ,etc

#define ➔ change / define keyword that will replace with constant / expression

- #define DEBUG
- #define MAX_VELOCITY 200
- #define circleArea(r) (3.1415*r*r)

```
#define __HAL_TIM_GET_COMPARE(__HANDLE__, __CHANNEL__) \
  (((__CHANNEL__) == TIM_CHANNEL_1) ? ((__HANDLE__)->Instance->CCR1) :\
   ((__CHANNEL__) == TIM_CHANNEL_2) ? ((__HANDLE__)->Instance->CCR2) :\
   ((__CHANNEL__) == TIM_CHANNEL_3) ? ((__HANDLE__)->Instance->CCR3) :\
   ((__HANDLE__)->Instance->CCR4))
```

operator

Precedence	Operator	Description	Associativity
1	++ -- () [] . -> (*type*){*list*}	Suffix/postfix increment and decrement Function call Array subscripting Structure and union member access Structure and union member access through pointer Compound literal(C99)	Left-to-right
2	++ -- + - ! ~ (*type*) * & sizeof _Alignof	Prefix increment and decrement Unary plus and minus Logical NOT and bitwise NOT Cast Indirection (dereference) Address-of Size-of Alignment requirement(C11)	Right-to-left
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ -	Addition and subtraction	
5	<< >>	Bitwise left shift and right shift	
6	< <= > >=	For relational operators < and ≤ respectively For relational operators > and ≥ respectively	
7	== !=	For relational = and ≠ respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10	|	Bitwise OR (inclusive or)	
11	&&	Logical AND	
12	||	Logical OR	
13	?:	Ternary conditional	Right-to-left
14	= += -= *= /= %= <<= >>= &= ^= |=	Simple assignment Assignment by sum and difference Assignment by product, quotient, and remainder Assignment by bitwise left shift and right shift Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right

arithmetic	bitwise	
+a	~a	bitwise NOT
-a	a & b	bitwise AND
a + b	a | b	bitwise OR
a - b	a ^ b	bitwise XOR (Use as bit toggle)
a * b	a << b	bitwise ShiftLeft
a / b	a >> b	bitwise ShiftRight
a % b (mod)		

comparison

a == b

a != b

a < b

a > b

a <= b

a >= b

logical

!a

a && b

a || b

Note:

logical and comraison always return true(1) or false(0)

Note2:

float a =0.1; //f = 0.100000001490116119384765625

double b =0.1; g = 0.1000000000000000055511151231257827021181583404541015625

(a==b) is false

assignment

```
a = b
a += b
a -= b
a *= b
a /= b
a %= b
a &= b
a |= b
a ^= b
a <<= b
a >>= b
```

increment
decrement

```
++a
--a
a++
a--
```

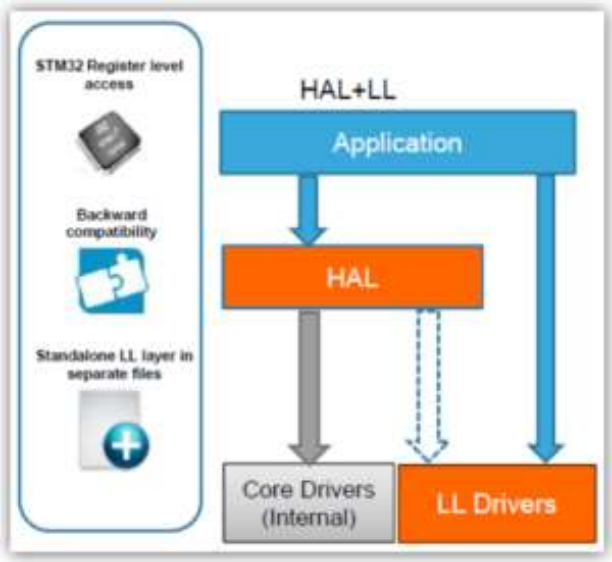
```
i = 1;
j = ++i;

i == 2, j == 2
```

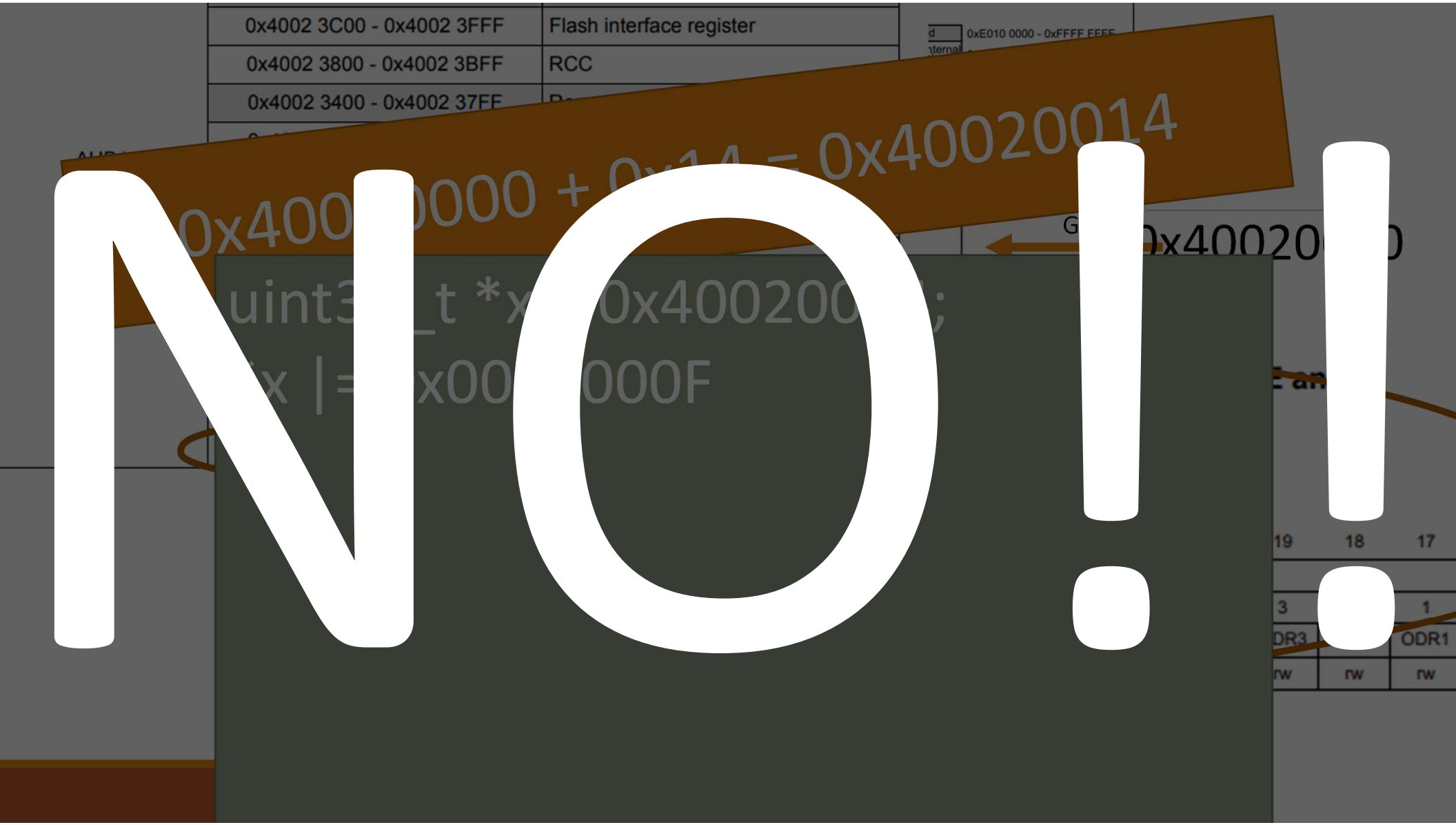
```
i = 1;
j = i++;

i == 2, j == 1
```

HAL - hardware abstraction layer

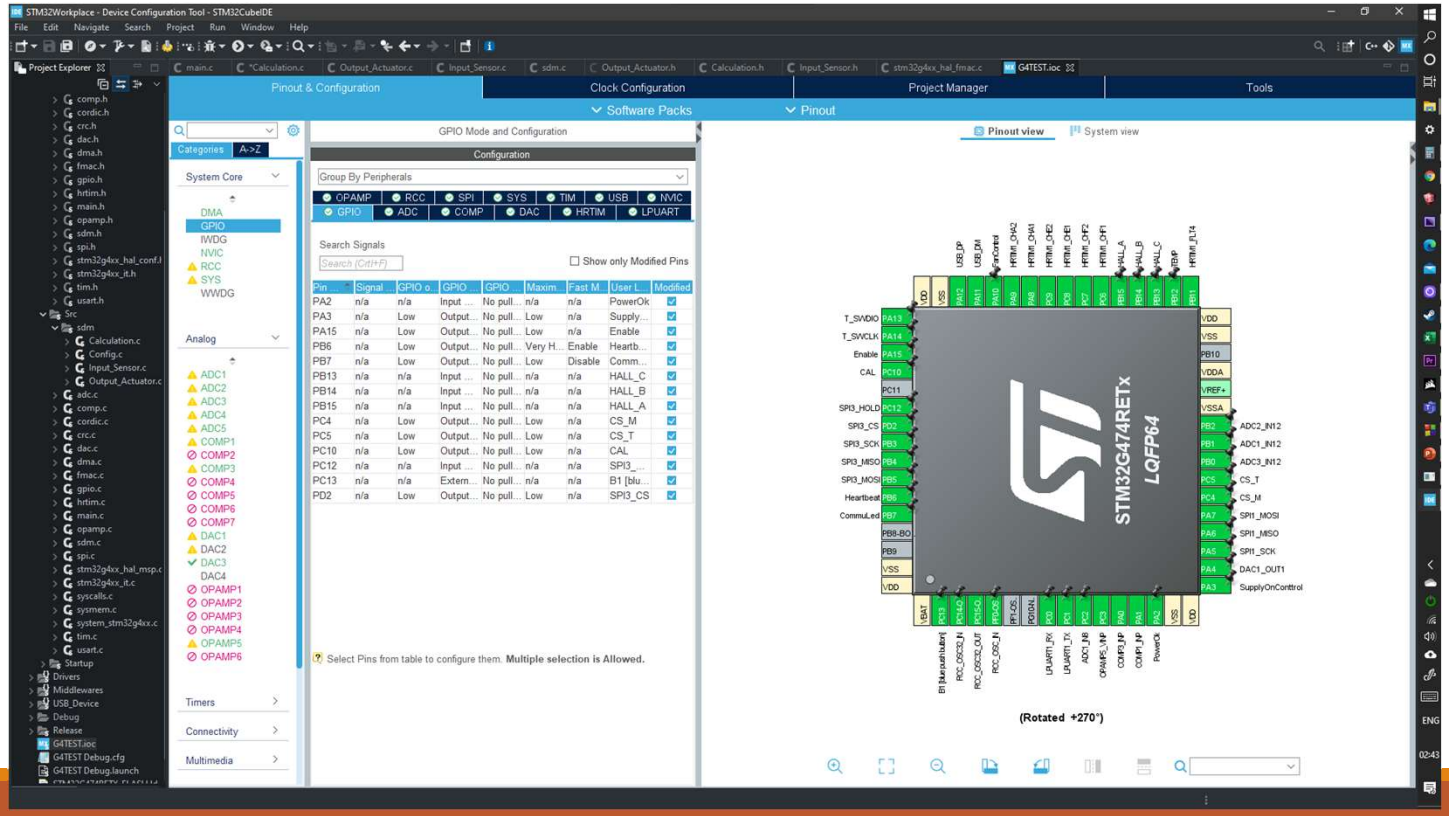
- implemented in software, between the physical hardware of a **computer** and the software that runs on that **computer**.

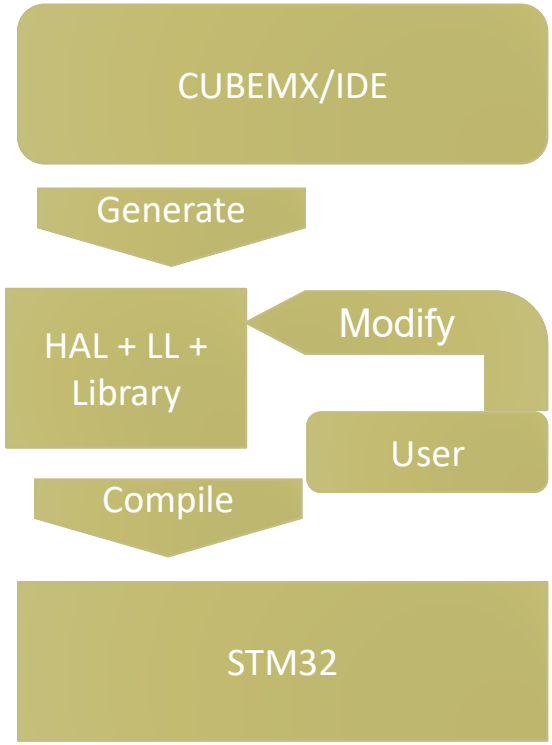
- In STM32 name of Libraries Layer implemented in software That connect between software and Hardware

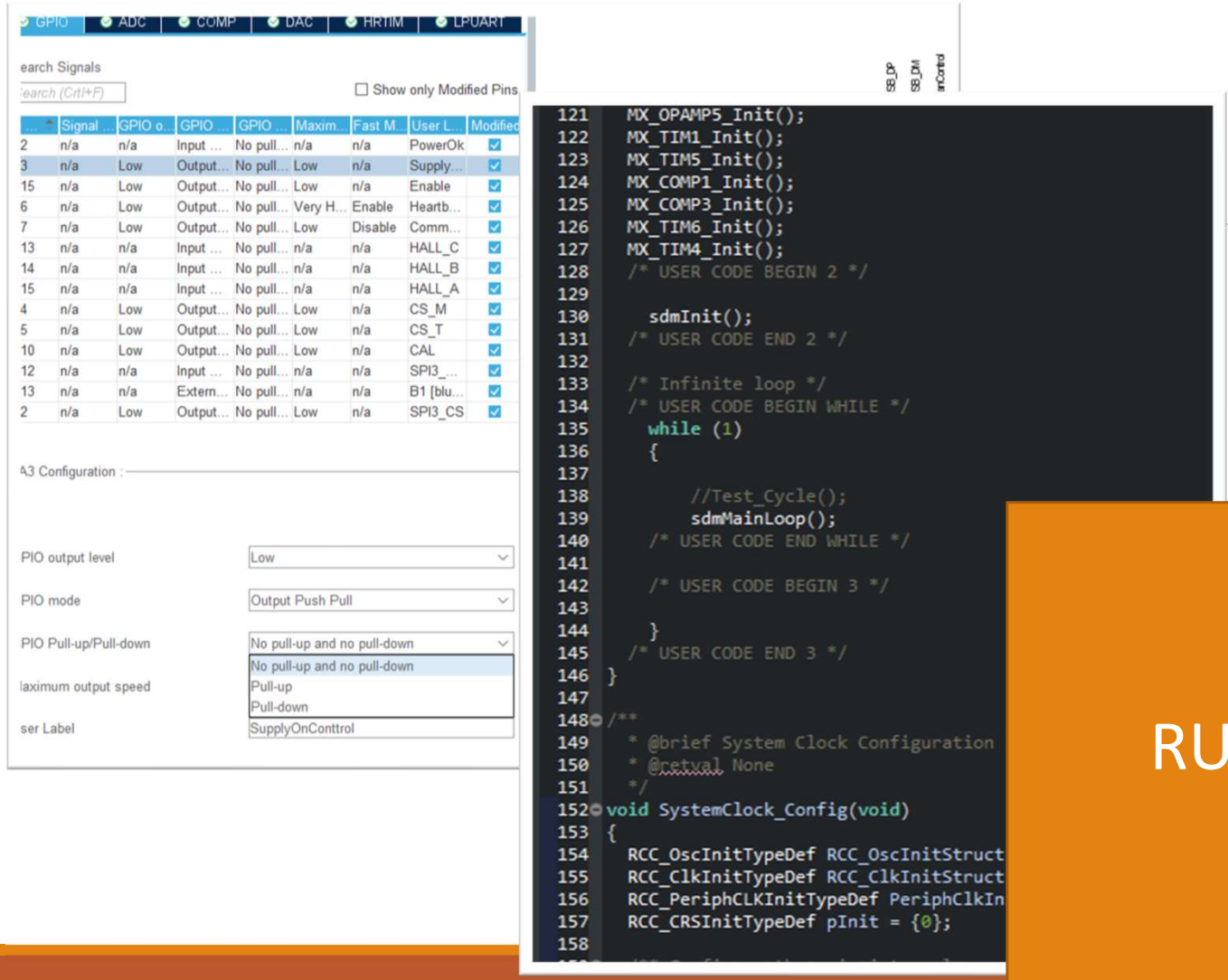



0x4002 3C00 - 0x4002 3FFF
Flash interface register
0x4002 3800 - 0x4002 3BFF
RCC
0x4002 3400 - 0x4002 37FF
0xE010 0000 - 0xFFFF FFFF
0x40020014
NO!!
19
18
17
3
1
ODR1
rw
rw
rw

STM32CubeMX/IDE






GPIO
ADC
COMP
DAC
HRTIM
LPUART
earch Signals
Show only Modified Pins
Signal ...
GPIO o...
GPIO ...
GPIO ...
Maxim...
Fast M...
User L...
Modified
PowerOk
Supply...
Enable
Heartb...
Comm...
HALL_C
HALL_B
HALL_A
CS_M
CS_T
CAL
SPI3_...
B1 [blu...
SPI3_CS
A3 Configuration :
PIO output level
Low
PIO mode
Output Push Pull
PIO Pull-up/Pull-down
No pull-up and no pull-down
Pull-up
Pull-down
laximum output speed
ser Label
SupplyOnConttrol
MX_OPAMP5_Init();
MX_TIM1_Init();
MX_TIM5_Init();
MX_COMP1_Init();
MX_COMP3_Init();
MX_TIM6_Init();
MX_TIM4_Init();
/* USER CODE BEGIN 2 */
sdmInit();
/* USER CODE END 2 */
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
//Test_Cycle();
sdmMainLoop();
/* USER CODE END WHILE */
/* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
}
/**
* @brief System Clock Configuration
* @retval None
*/
void SystemClock_Config(void)
{
RCC_OscInitTypeDef RCC_OscInitStruct
RCC_ClkInitTypeDef RCC_ClkInitStruct
RCC_PeriphCLKInitTypeDef PeriphClkIn
RCC_CRSInitTypeDef pInit = {0};

RUN + DEBUG

Example HAL

HAL_GetTick();

- คือ millis(); ใน Arduino , us_ticker_read() ใน mbed เทอมที่แล้ว
- return เวลา นับจากเริ่มโปรแกรม ในหน่วย ms ชนิด uint32_t

HAL_GPIO_WritePin(GPIOx, GPIO_Pin, PinState);

- เขียน output GPIO
- ex >> HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET); //PA4 set เป็น high

HAL_GPIO_ReadPin(GPIOx, GPIO_Pin)

- อ่านค่าlogic บนpin ปัจจุบัน

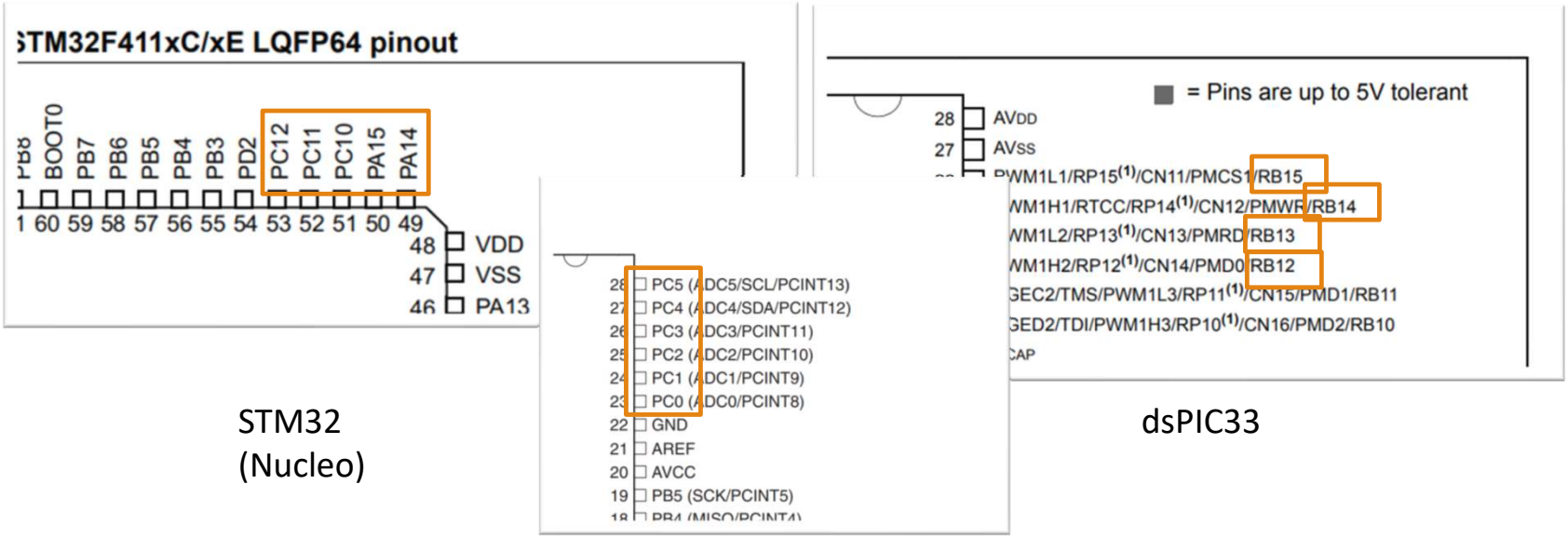
HAL_GPIO_TogglePin(GPIOx, GPIO_Pin);

- toggle logic pin ปัจจุบัน

GPIO ?

~~D0 D1 D2 A0 A1 A2 ???~~

GPIO - General Propose Input Output << Peripheral



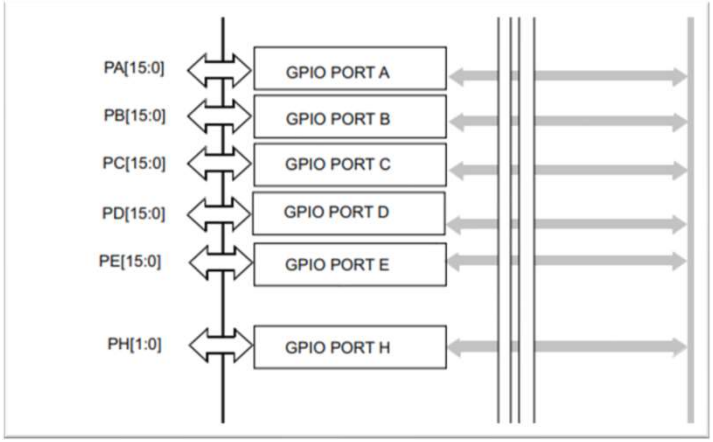
STM32
(Nucleo)

dsPIC33

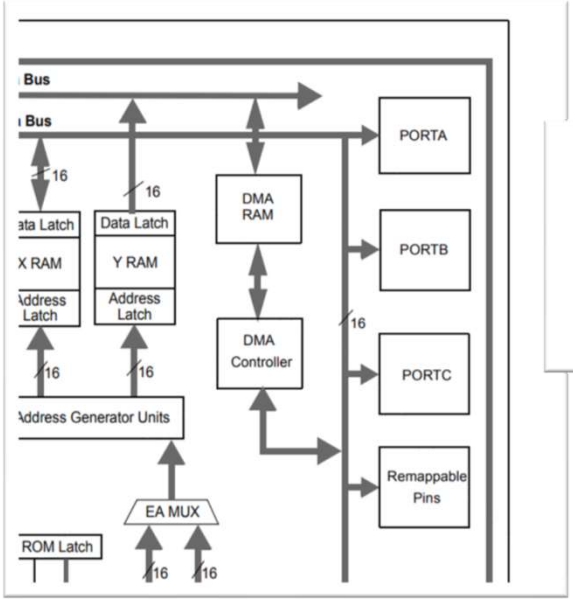
AVR
(Arduino)

GPIO

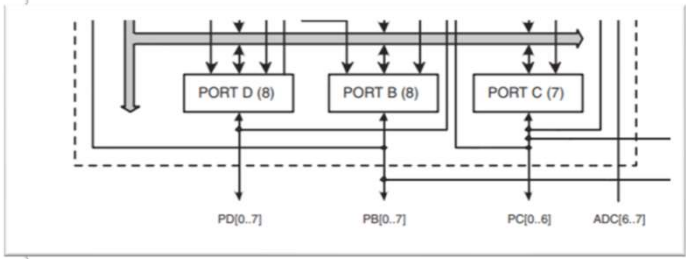
- PORT / GPIO



STM32
(Nucleo)



dsPIC33



AVR
(Arduino)

STM32 GPIO

Name by Port and pin

- PA4
- PC15
- PF0

Table 8. STM32F411xC/xE pin definitions (continued)

Pin number					Pin name (function after reset)(1)	Pin type	I/O structure	Notes	Alternate functions	Additional functions
UFQFPN48	LQFP64	WLCSP49	LQFP100	UFBGA100						
10	14	F6	23	L2	PA0-WKUP	I/O	TC	(5)	TIM2_CH1/TIM2_ET, TIM5_CH1, USART2_CTS, EVENTOUT	ADC1_0, WKUP1
11	15	G7	24	M2	PA1	I/O	FT	-	TIM2_CH2, TIM5_CH2, SPI4_MOSI/I2S4_SD, USART2_RTS, EVENTOUT	ADC1_1
12	16	E5	25	K3	PA2	I/O	FT	-	TIM2_CH3, TIM5_CH3, TIM9_CH1, I2S2_CKIN, USART2_TX,	ADC1_2

https://www.st.com/resource/en/datasheet/stm32f411ce.pdf
Page 40

Table 7. Legend/abbreviations used in the pinout table

Name	Abbreviation	Definition
Pin name	Unless otherwise specified in brackets below the pin name, the pin function during and after [illegible] as the actual pin name	
Pin type	S	Supply pin
	I	Input only pin
	I/O	Input/ output pin
I/O structur	FT	5 V tolerant I/O
	TC	Standard 3.3 V I/O
	B	Dedicated BOOT0 pin
	NRST	Bidirectional reset pin with embedded weak pull-up resistor
Notes	Unless otherwise specified by a note, all I/Os are set as floating inputs during and after reset	
Alternate functions	Functions selected through GPIOx_AFR registers	
Additional functions	Functions directly selected/enabled through peripheral registers	

CANNOT Output 5V !!!

STM32

Table 11. Voltage characteristics

Symbol	Ratings	Min	Max	
$V_{DD}-V_{SS}$	External main supply voltage (including V_{DDA}, V_{DD} and V_{BAT})(1)	–0.3	4.0	
V_{IN}	Input voltage on FT and TC pins(2)	V_{SS}–0.3	V_{DD}+4.	
	Input voltage on any other pin	V_{SS}–0.3	4.0	
	Input voltage for BOOT0	V_{SS}	9.0	
$\lvert\Delta V_{DDx}\rvert$	Variations between different V_{DD} power pins	-	50	mV
$\lvert V_{SSX}-V_{SS}\rvert$	Variations between all the different ground pins	-	50	
$V_{ESD(HBM)}$	Electrostatic discharge voltage (human body model)	see *Section 6.3.14: Absolute maximum ratings (electrical sensitivity)*		

1. All main power (V_{DD}, V_{DDA}) and ground (V_{SS}, V_{SSA}) pins must always be connected to the external power supply, in the permitted range.
2. V_{IN} maximum value must always be respected. Refer to *Table 12* for the values of the maximum allowed injected current.

Table 12. Current characteristics

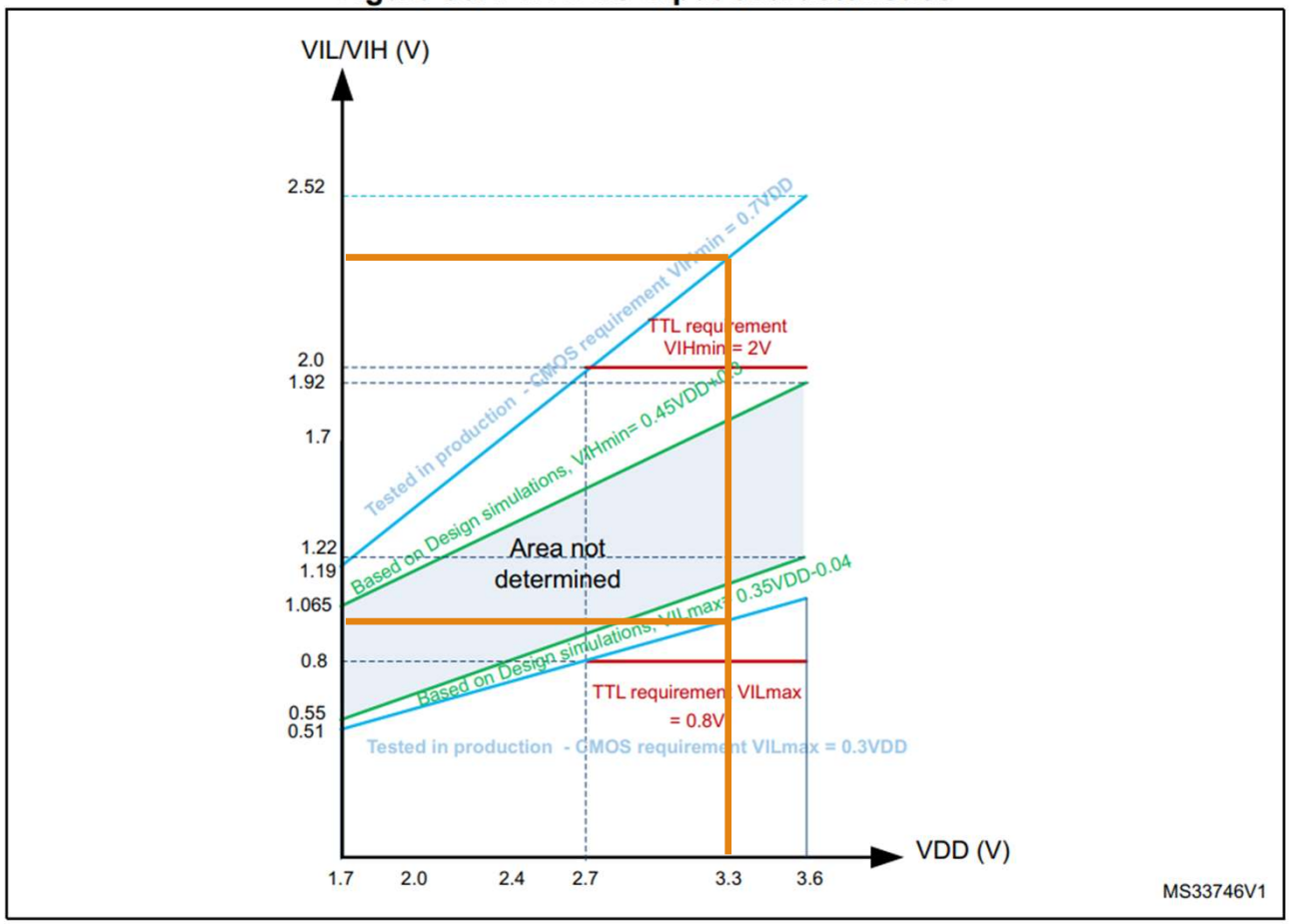
Symbol	Ratings	Max.	Unit
ΣI_{VDD}	Total current into sum of all V_{DD_x} power lines (source)(1)	160	mA
ΣI_{VSS}	Total current out of sum of all V_{SS_x} ground lines (sink)(1)	-160	
I_{VDD}	Maximum current into each V_{DD_x} power line (source)(1)	100	
I_{VSS}	Maximum current out of each V_{SS_x} ground line (sink)(1)	-100	
I_{IO}	Output current sunk by any I/O and control pin	25	
	Output current sourced by any I/O and control pin	-25	
ΣI_{IO}	Total output current sunk by sum of all I/O and control pins (2)	120	
	Total output current sourced by sum of all I/Os and control pins(2)	-120	
$I_{INJ(PIN)}$ (3)	Injected current on FT and TC pins (4)	–5/+0	
	Injected current on NRST and B pins (4)		
$\Sigma I_{INJ(PIN)}$	Total injected current (sum of all I/O and control pins)(5)	±25	

STM32

Maximum current out of VSS pin	300 mA
Maximum current into VDD pin(2)	250 mA
Maximum current sourced/sunk by any 2x I/O pin(3)	8 mA
Maximum current sourced/sunk by any 4x I/O pin(3)	15 mA
Maximum current sourced/sunk by any 8x I/O pin(3)	25 mA
Maximum current sunk by all ports	200 mA

dsPIC33

Figure 30. FT/TC I/O input characteristics



STM32

STM32

Output driving current

The GPIOs (general purpose input/outputs) can sink or source up to ±8 mA, and sink or source up to ±20 mA (with a relaxed V_{OL}/V_{OH}) except PC13, PC14 and PC15 which can sink or source up to ±3mA. When using the PC13 to PC15 GPIOs in output mode, the speed should not exceed 2 MHz with a maximum load of 30 pF.

In the user application the number of I/O pins which can drive current must be limited to respect the absolute m

- The sum of the cu consumption of th ΣI_{VDD} (see *Table*
- The sum of the cu consumption of th ΣI_{VSS} (see *Table*

Table 54. Output voltage characteristics

Symbol	Parameter	Conditions	Min	Max	Unit
V_{OL}(1)	Output low level voltage for an I/O pin	CMOS port(2) I_{IO} = +8 mA 2.7 V ≤V_{DD} ≤3.6 V	-	0.4	V
V_{OH}(3)	Output high level voltage for an I/O pin		V_{DD}–0.4	-	
V_{OL}(1)	Output low level voltage for an I/O pin	TTL port(2) I_{IO} =+8 mA 2.7 V ≤V_{DD} ≤3.6 V	-	0.4	V
V_{OH}(3)	Output high level voltage for an I/O pin		2.4	-	
V_{OL}(1)	Output low level voltage for an I/O pin	I_{IO} = +20 mA 2.7 V ≤V_{DD} ≤3.6 V	-	1.3(4)	V
V_{OH}(3)	Output high level voltage for an I/O pin		V_{DD}–1.3(4)	-	

Conclusion

To use GPIO (And Microcontroller)

- Voltage
- Current
- Type
- Etc.

Let try Blink LED 1Hz , But in HAL

Keyword

- HAL_GPIO_WritePin(port,pin);
- HAL_Delay()
- while(1)