ARC IoTDK

- Lab 6 -

Source

- **Guide Video & Lab Video :**
  https://www.youtube.com/playlist?list=PLtOgiVU90A6_jJ7KYYybvQ1WFFaivElLi

- **Example Code :**
  https://github.com/waynelinbo/LAB6_ARC_IoTDK_Example_Code

Content

- Introduction
- GPIO
- TIMER & INTERRUPT
- PWM

Introduction

ARC

Link：
1. https://www.synopsys.com/dw/doc.php/ds/cc/dw-processor-solutions.pdf
2. https://embarc.org/embarc_osp/doc/build/html/arc/arc.html

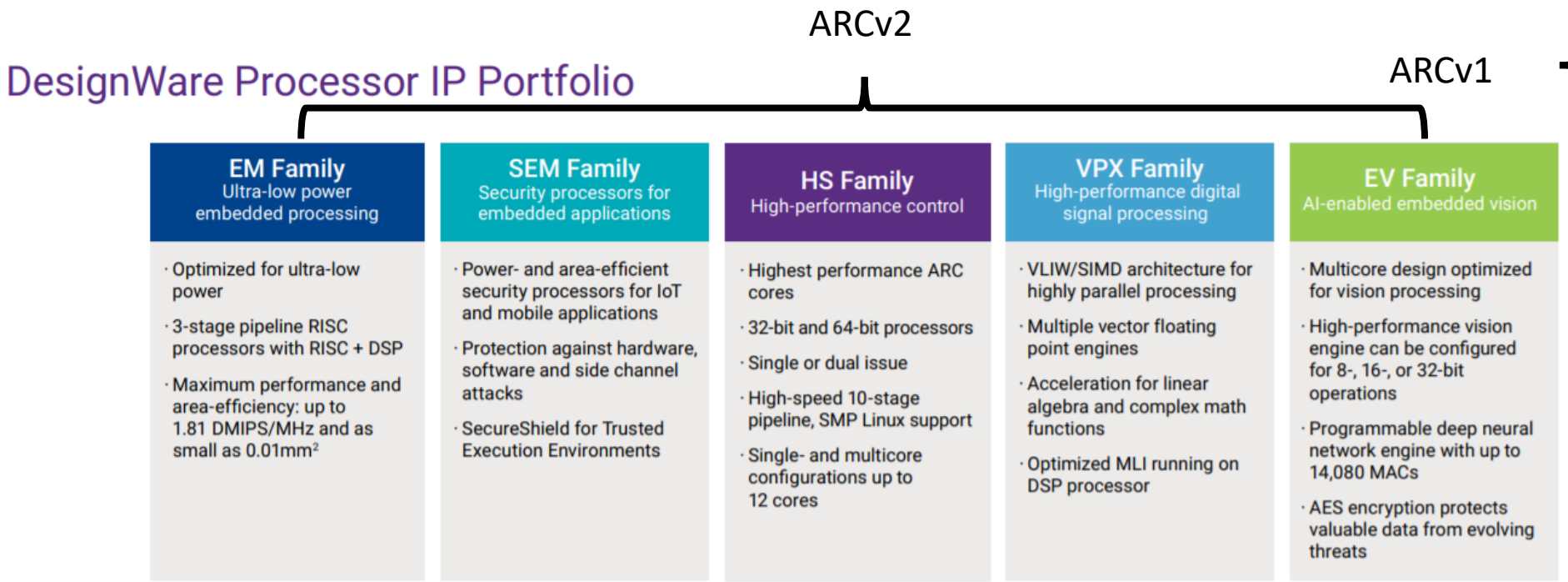
- Synopsys (新思科技) 開發的處理器架構
- 有自己的 ISA
- 多用於嵌入式應用

ARCv2

高端嵌入式、支持 Linux	HS Family
超低功號控制+DSP	EM Family

ARCv1

中高端嵌入式	700 Family
通用嵌入式 & DSP	600 Family
音頻處理	AS200 Family



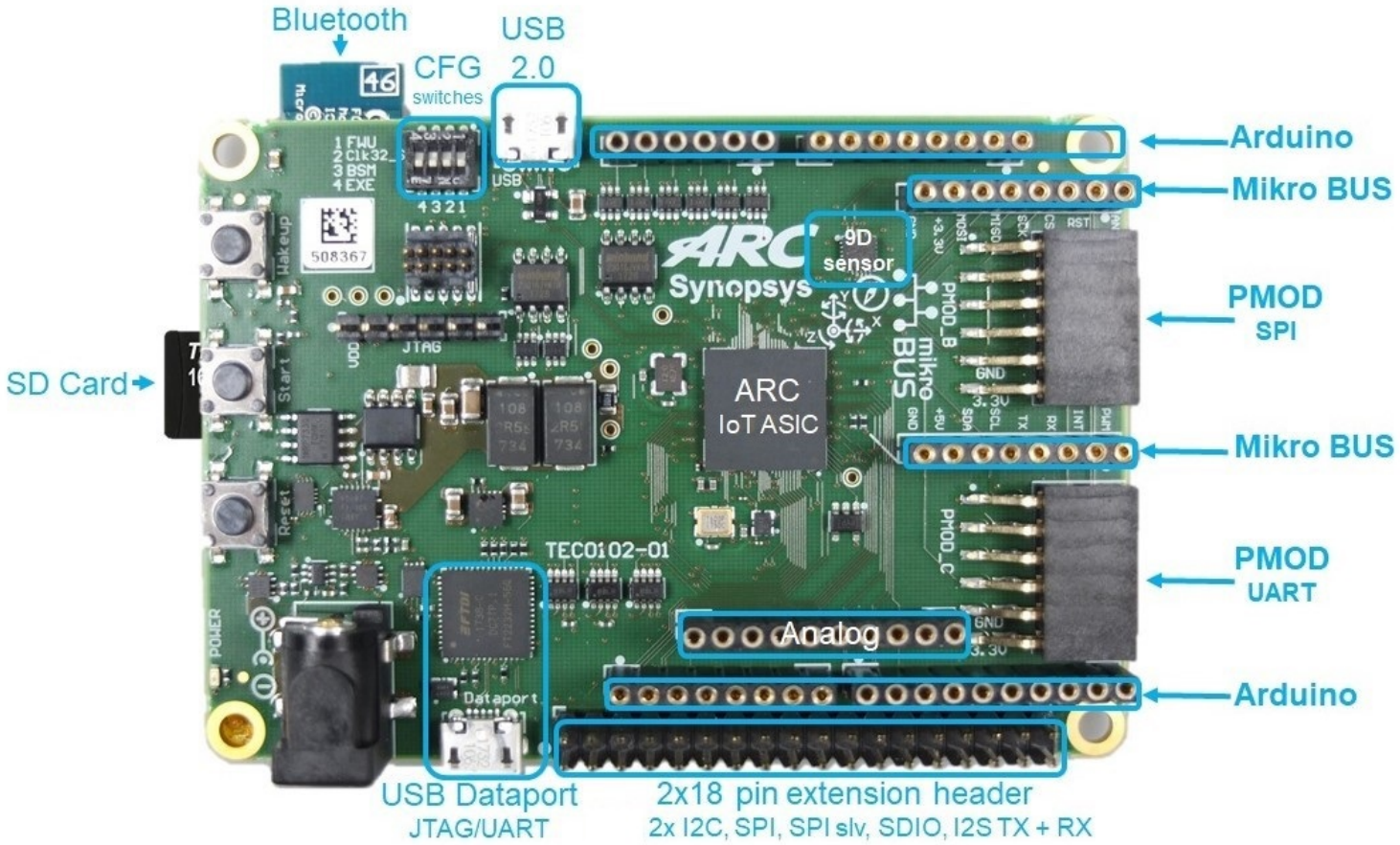
IoTDK

Link :
https://embarc.org/embarc_osp/doc/build/html/board/iotdk.html

- EM 系列的一個產品

處理器 : EM9D @ 144MHz

- Interfaces
  - USB2.0 OTG
  - USB Data port (JTAG/UART)
  - Micro SD Card
  - 9D Sensor (Invensense MPU-9250)
  - BTLE module (Microchip RN-4020)
  - ADC (16 channels)
  - JTAG
  - PMIC with dynamic voltage control

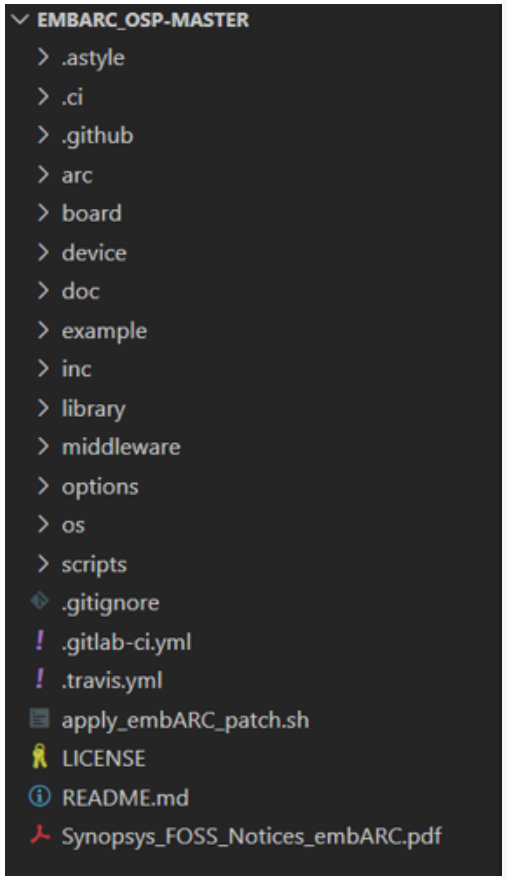


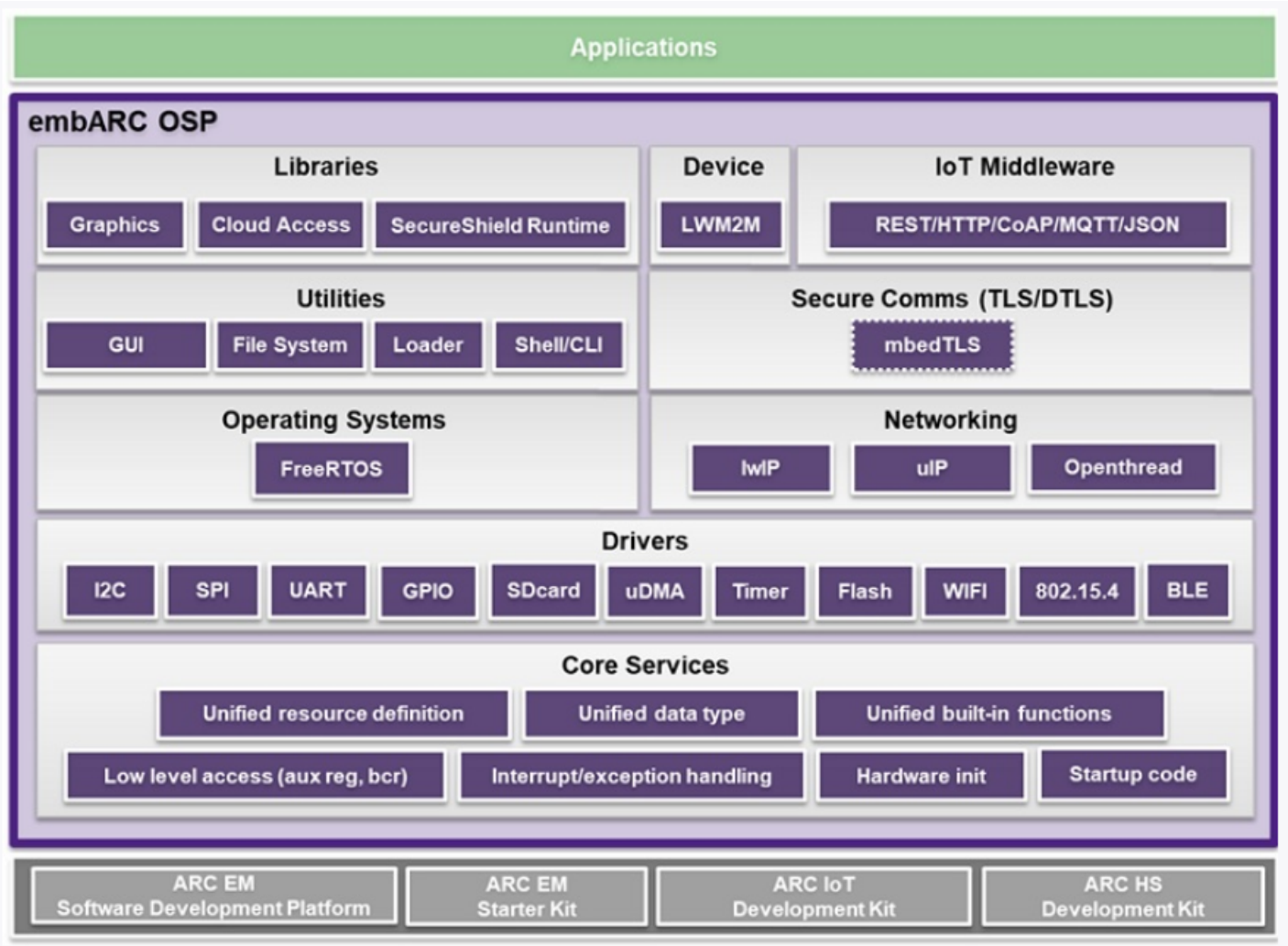
embARC OSP (Open Software Platform)

Link :
https://github.com/foss-for-synopsys-dwc-arc-processors/embarc_osp

- 由 C code 為基底寫成
- 板子裡運作的東西
- 可以寫應用程式 (C code) 透過OSP, link起來跑在不同的硬體上

Application
1. C code (main.c)
2. makefile





GNU Toolchain

GNU開發工具是Linux作業系統、嵌入式系統、與自由及開放原始碼軟體社群中，最常見、使用得最廣泛的程式開發工具，可以協助程式設計人員完成**從原始程式碼到二進制機械碼**時各項煩瑣的工作，能夠大幅提昇程式的執行效率。不但支援**多種程式語言、多種作業系統**，也可以產生**多種不同處理器的執行檔**，是**跨平台**程式開發和移植的最佳工具。

GNU開發工具是由GNU計劃 (GNU Project) 中的**數個程式開發工具所集合而成**，可以完成編譯、組譯和連結等步驟，還有自動化設定、自動化編譯、除錯工具等功能。

GNU make	自動化編譯工具
GNU Compiler Collection (GCC)	編譯器套件
GNU Binutils	二進制工具
GNU Debugger (GDB)	除錯器
GNU Build System	自動化程式建構系統

$ make run
讀取makefike定義的shell指令並依其時間相依性規則執行

VSCode

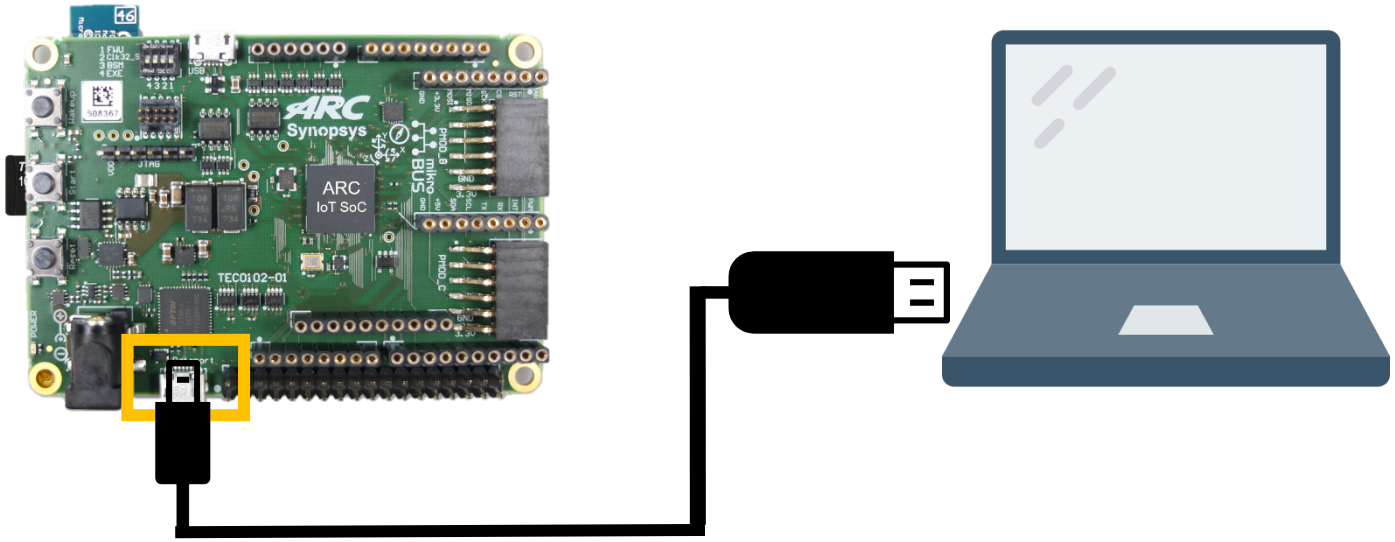
由微軟開發，同時支援Windows 、 Linux和macOS等操作系統的免費**程式碼編輯器**。
支援多種程式語言，整合終端，可以在編輯器中執行指令碼、編譯軟體、除錯指令碼、設定斷點、做版本管理。
支援用戶個性化組態，同時還在編輯器中內建了擴充程式管理的功能。

1. 擁有所有文本編輯器的功能
2. 可任意安裝擴充套件進行編譯執行 (文本編輯器做不到)

2016年正式發布

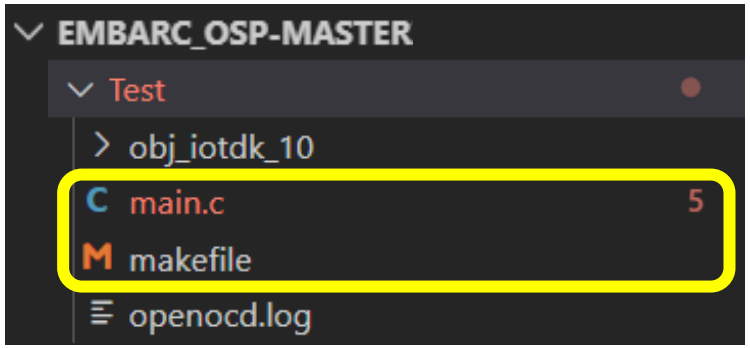
Putty

Putty是一款**整合虛擬終端**、系統控制台和網路檔案傳輸為一體的自由及開放原始碼的程式。
它支援多種網路協定，包括SCP，SSH，Telnet，rlogin和原始的通訊端連接。
它也可以連接到序列埠(Serial Port)。



TO DO

File structure



1. main.c

```c
#include "embARC.h"
#include "embARC_debug.h"


int main(void)
{
    // ...
    return E_SYS
}
```

2. makefile

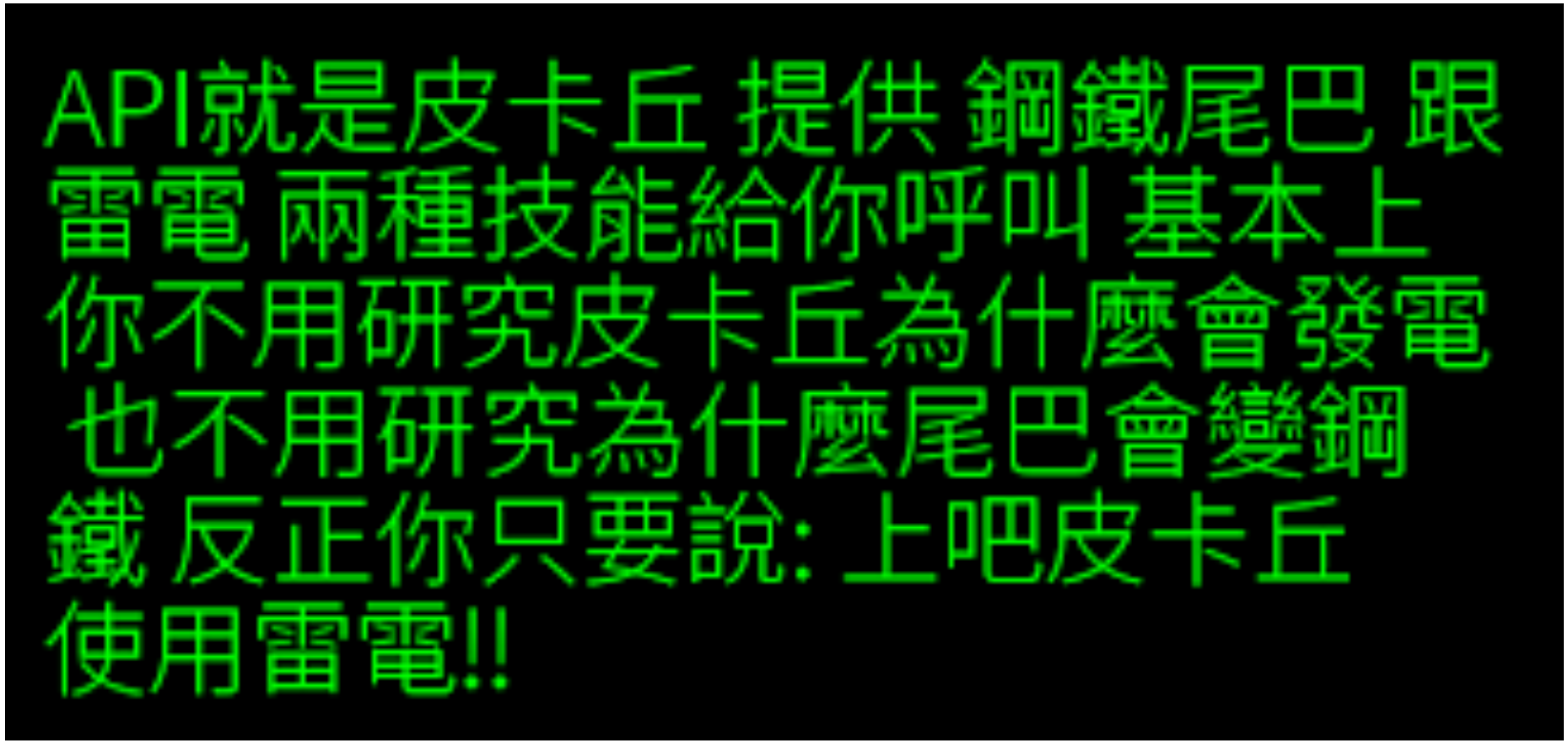
```
# Application name
APPL ?= blinky

BOARD ?=iotdk

TOOLCHAIN ?= gnu

OLEVLE ?=O0

#                    EMBARC_ROOT = Where is EMBARC_OSP-MASTER
# root dir of embARC
#
EMBARC_ROOT = ../

MID_SEL = common

# application source dirs
APPL_CSRC_DIR = .
APPL_ASMSRC_DIR = .

# application include dirs
APPL_INC_DIR = .

# include current project makefile
COMMON_COMPILE_PREREQUISITES += makefile

### Options above must be added before include options.mk ###
# include key embARC build system makefile
include $(EMBARC_ROOT)/options/options.mk
```

Must change if you change your File structure

3. Terminal command

```
終端機   問題   輸出   偵錯主控台

Windows PowerShell
Copyright (C) Microsoft Corporation. 著作權所有，並保留一切權利。

請嘗試新的跨平台 PowerShell https://aka.ms/pscore6

PS D:\User\Desktop\嘎啦嘎拉\embarc_osp-master\embarc_osp-master> cd Test
PS D:\User\Desktop\嘎啦嘎拉\embarc_osp-master\embarc_osp-master\Test> make run
```

API



延伸閱讀 **(API vs ABI)** : https://www.itread01.com/content/1545181580.html

Useful Functions

EMBARC_PRINTF() ➔ C printf

```
void board_delay_ms(uint32_t ms, uint8_t os_compat)
```

Ex :

delay 1 second ➔ board_delay_ms(1000, 0);

```
uint64_t board_get_cur_us(void)
```

Ex :

```
uint64_t  now_us = board_get_cur_us();
// the unit of “now_us” is us
```

GPIO

GPIO - Introduction

General-Purpose Input/Output，
a generic pin on an integrated circuit whose behavior, including whether it is an input or output pin, can be controlled by the user at run time.

1. GPIO常見於開發版邊緣，以針腳 (Pin) 的形式呈現，這些針腳即是開發版與外界溝通的重要橋樑
2. 具有彈性且可以藉由軟體控制 (software-controlled) 的 數位訊號
3. 可以由外界給予 1/0 (輸入)，或者給予外界 1/0 (輸出)

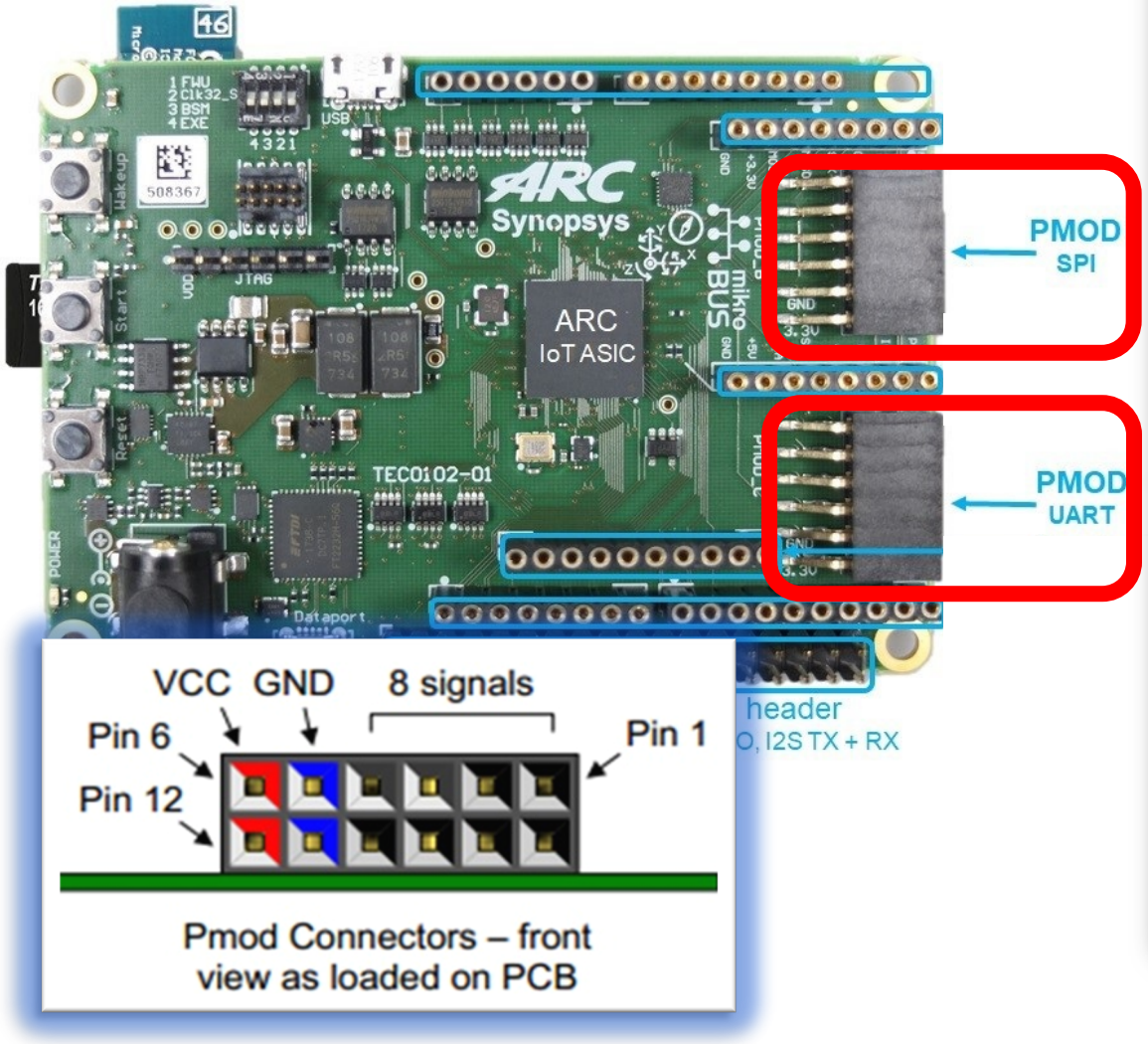
ARC IoTDK GPIO Pins :

1 pin = 1 bit

- 44 bit GPIO (4 x 8-bit + 3 x 4-bit)

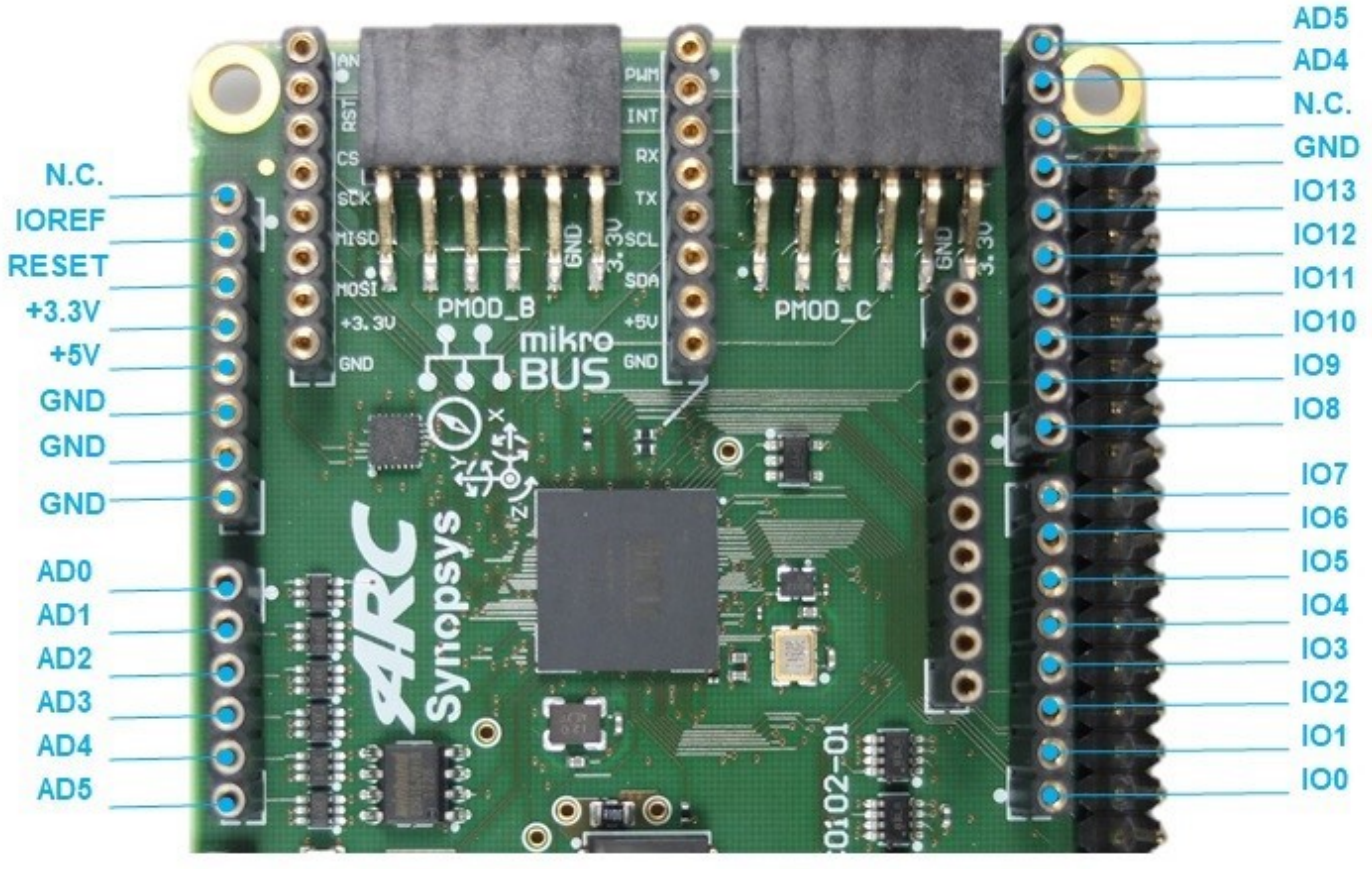
gpio8b_0
gpio8b_1
gpio8b_2
gpio8b_3
gpio4b_0
gpio4b_1
gpio4b_2

GPIO - PMOD Pins



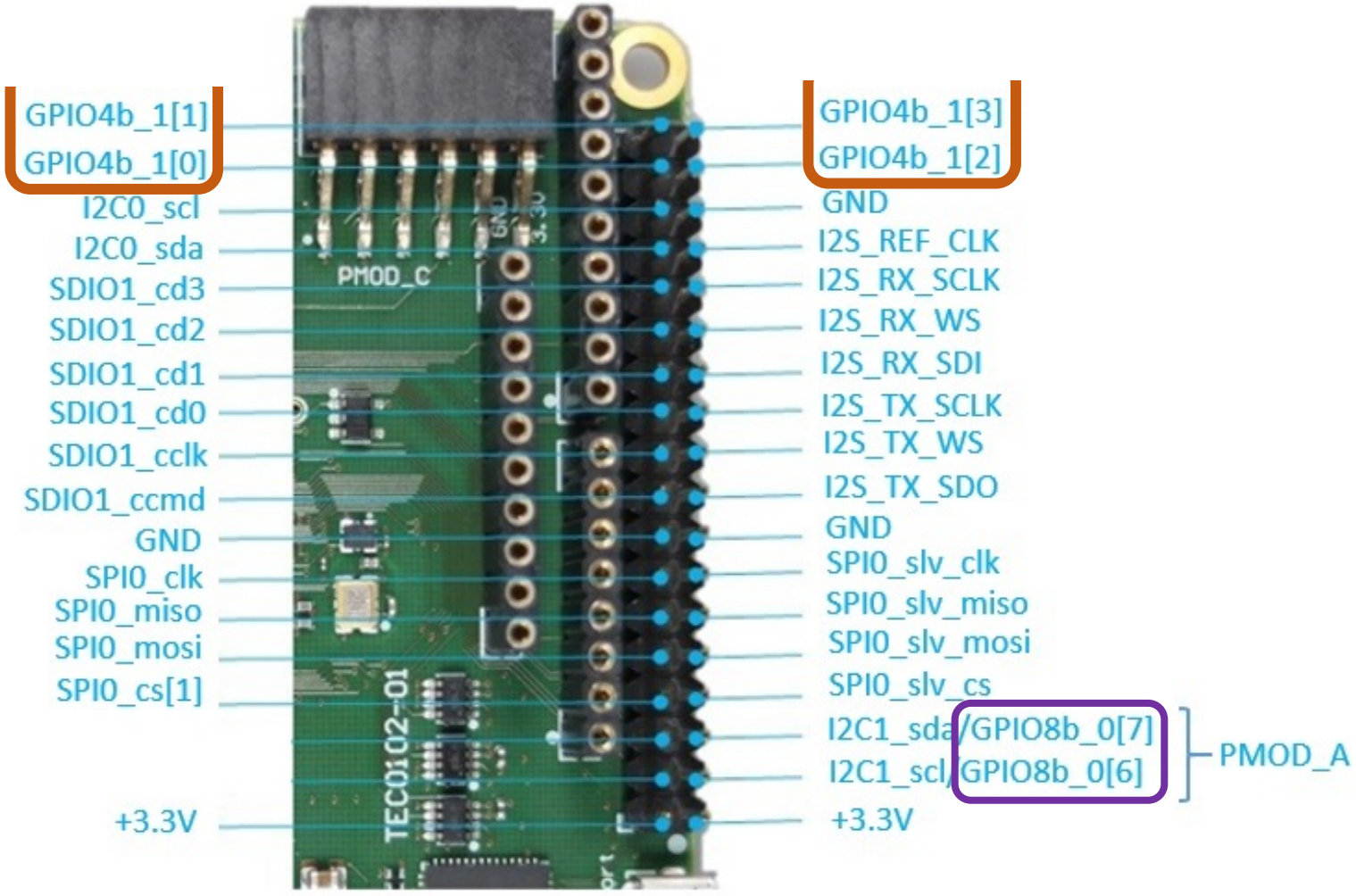
POMD_B			POMD_C		
Pin	**GPIO**	**SPI**	**Pin**	**GPIO**	**UART**
B1	gpio8b_0[0]	SPI1_CS_N[0]	C1	gpio8b_1[0]	UART1_CTS
B2	gpio8b_0[1]	SPI1_MOSI	C2	gpio8b_1[1]	UART1_TXD
B3	gpio8b_0[2]	SPI1_MISO	C3	gpio8b_1[2]	UART1_RXD
B4	gpio8b_0[3]	SPI1_CLK	C4	gpio8b_1[3]	UART1_RTS
B5	GND	GND	C5	GND	GND
B6	3V3	3V3	C6	3V3	3V3
B7	gpio8b_0[4]	gpio8b_0[4]	C7	gpio8b_1[4]	gpio8b_1[4]
B8	gpio8b_0[5]	gpio8b_0[5]	C8	gpio8b_1[5]	gpio8b_1[5]
B9	N.C	N.C	C9	N.C	N.C
B10	N.C	N.C	C10	N.C	N.C
B11	GND	GND	C11	GND	GND
B12	3V3	3V3	C12	3V3	3V3

GPIO - Arduino Pins



Pin	MUX_bitfield	I/O-0	I/O-1	I/O-2
AD0	Bit 10	ADC IN0/gpio8b_3[7]	·	·
AD1	Bit 11	ADC IN1/gpio8b_3[6]	·	·
AD2	Bit 12	ADC IN2/gpio8b_3[5]	·	·
AD3	Bit 13	ADC IN3/gpio8b_3[4]	·	·
AD4	Bit 8/14	ADC IN4/gpio8b_3[3]	i2c2_sda	·
AD5	Bit 8/15	ADC IN5/gpio8b_3[2]	i2c2_scl	·
IO0	Bit 0	gpio4b_2[0]	uart2_rxd	·
IO1	Bit 0	gpio4b_2[1]	uart2_txd	·
IO2	·	gpio4b_2[2]	·	·
IO3	Bit 2	gpio4b_2[3]	·	pwm0
IO4	·	gpio8b_2[0]	·	·
IO5	Bit 3	gpio8b_2[1]	·	pwm1
IO6	Bit 4	gpio8b_2[2]	·	pwm2
IO7	·	gpio8b_2[3]	·	·
IO8	·	gpio8b_2[4]	·	·
IO9	Bit 5	gpio8b_2[5]	·	pwm3
IO10	Bit 1/6	gpio8b_2[6]	spi2_cs_n	pwm4
IO11	Bit 1/7	gpio8b_2[7]	spi2_mosi	pwm5
IO12	Bit 1	gpio8b_3[0]	spi2_miso	gpio8b_3[0]
IO13	Bit 1	gpio8b_3[1]	spi2_clk	gpio8b_3[1]

GPIO - Extension Header Pins

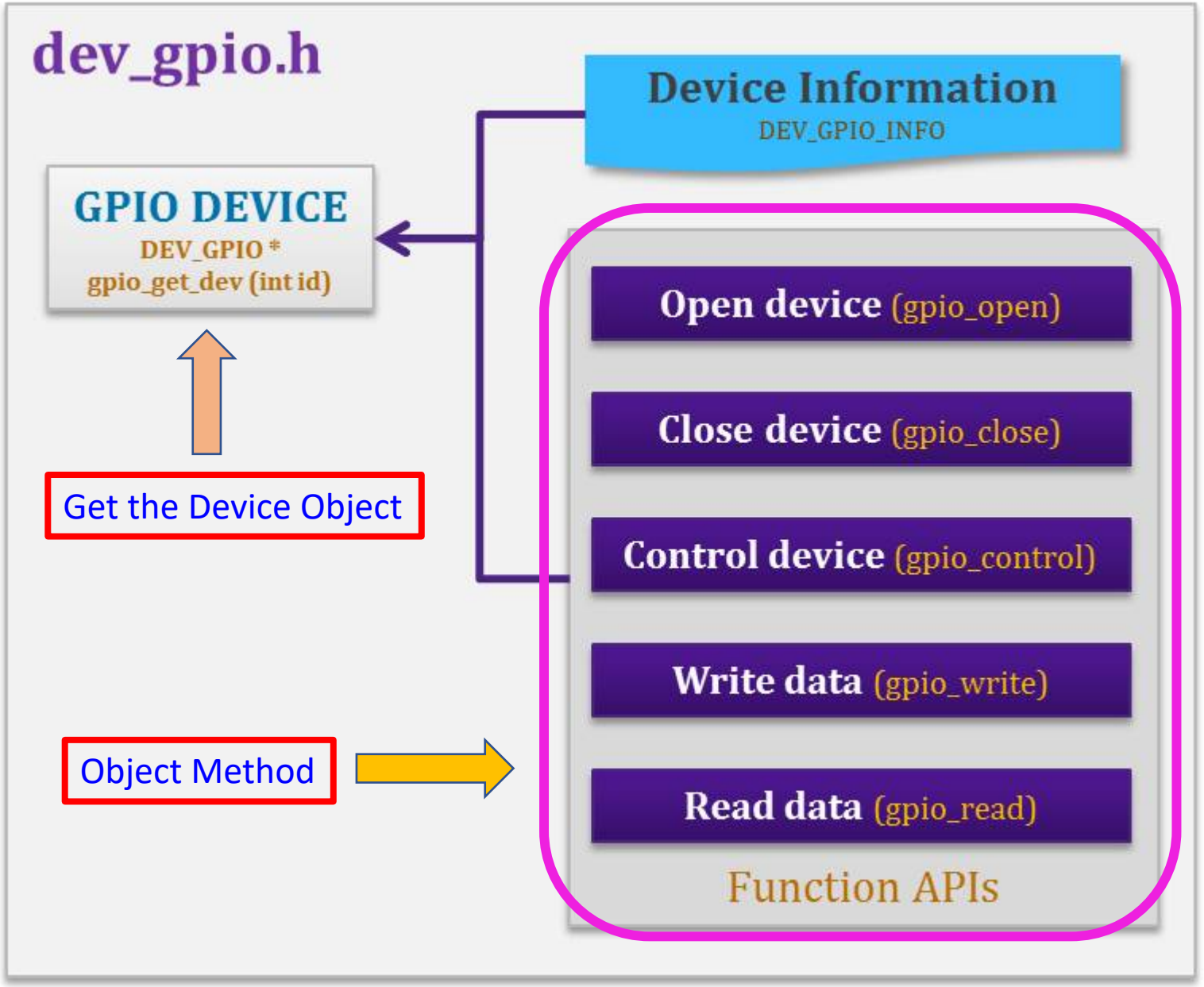


GPIO - API

Link :
https://embarc.org/embarc_osp/doc/build/html/device/gpio.html

```
DEV_GPIO_PTR gpio_get_dev(int32_t gpio_id);
```

```
typedef struct dev_gpio {
    DEV_GPIO_INFO gpio_info;        /*!< gpio device informat
    int32_t (*gpio_open) (uint32_t dir);    /*!< open gpio de
    int32_t (*gpio_close) (void);       /*!< close gpio devic
    int32_t (*gpio_control) (uint32_t ctrl_cmd, void *param);
    int32_t (*gpio_write) (uint32_t val, uint32_t mask);    /
    int32_t (*gpio_read) (uint32_t *val, uint32_t mask);    /
} DEV_GPIO, * DEV_GPIO_PTR;
```



GPIO - API (gpio_get_dev)

```
DEV_GPIO_PTR gpio_get_dev(int32_t gpio_id);
```

ID :

```
#define DFSS_GPIO_8B0_ID        4   /* GPIO 8 ID macro (io_gpio_8b0)
#define DFSS_GPIO_8B1_ID        5   /* GPIO 9 ID macro (io_gpio_8b1)
#define DFSS_GPIO_8B2_ID        6   /* GPIO 10 ID macro (io_gpio_8b2)
#define DFSS_GPIO_8B3_ID        7   /* GPIO 11 ID macro (io_gpio_8b3)
#define DFSS_GPIO_4B0_ID        8   /* GPIO 4 ID macro (io_gpio_4b0)
#define DFSS_GPIO_4B1_ID        9   /* GPIO 5 ID macro (io_gpio_4b1)
#define DFSS_GPIO_4B2_ID        10  /* GPIO 6 ID macro (io_gpio_4b2)
```

- gpio8b_0
- gpio8b_1
- gpio8b_2
- gpio8b_3
- gpio4b_0
- gpio4b_1
- gpio4b_2

return :

```
typedef struct dev_gpio {
    DEV_GPIO_INFO gpio_info;          /*!< gpio device informat
    int32_t (*gpio_open) (uint32_t dir);     /*!< open gpio de
    int32_t (*gpio_close) (void);        /*!< close gpio devic
    int32_t (*gpio_control) (uint32_t ctrl_cmd, void *param);
    int32_t (*gpio_write) (uint32_t val, uint32_t mask);     /
    int32_t (*gpio_read) (uint32_t *val, uint32_t mask);     /
} DEV_GPIO, * DEV_GPIO_PTR;
```

EX :

```
DEV_GPIO_PTR gpio_4b2 = gpio_get_dev(DFSS_GPIO_4B2_ID);
```

```
475  DEV_GPIO_PTR gpio_get_dev(int32_t gpio_id)
476  {
477      static uint32_t install_flag = 0;
478
479      /* intall device objects */
480      if (install_flag == 0) {
481          install_flag = 1;
482          dfss_gpio_all_install();
483      }
484
485      switch (gpio_id) {
486  #if (USE_DFSS_GPIO_8B0)
487
488          case DFSS_GPIO_8B0_ID: return &gpio_0;
489  #endif
490  #if (USE_DFSS_GPIO_8B1)
491
492          case DFSS_GPIO_8B1_ID: return &gpio_1;
493  #endif
494  #if (USE_DFSS_GPIO_8B2)
495
496          case DFSS_GPIO_8B2_ID: return &gpio_2;
497  #endif
498  #if (USE_DFSS_GPIO_8B3)
499
500          case DFSS_GPIO_8B3_ID: return &gpio_3;
501  #endif
502  #if (USE_DFSS_GPIO_4B0)
503
504          case DFSS_GPIO_4B0_ID: return &gpio_4;
505  #endif
506  #if (USE_DFSS_GPIO_4B1)
507
508          case DFSS_GPIO_4B1_ID: return &gpio_5;
509  #endif
510  #if (USE_DFSS_GPIO_4B2)
511
512          case DFSS_GPIO_4B2_ID: return &gpio_6;
513  #endif
```

GPIO - API (gpio_open)

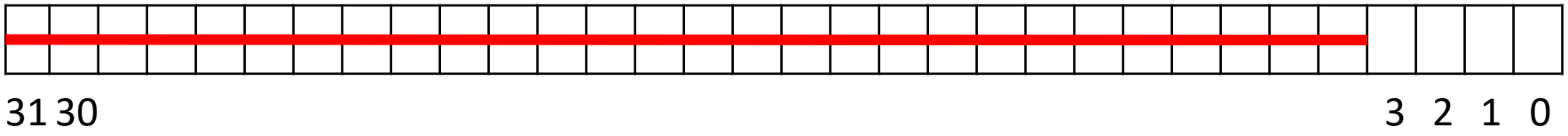
```
typedef struct dev_gpio {
    DEV_GPIO_INFO gpio_info;          /*!< gpio device informat
    int32_t (*gpio_open) (uint32_t dir);    /*!< open gpio de
    int32_t (*gpio_close) (void);         /*!< close gpio devic
    int32_t (*gpio_control) (uint32_t ctrl_cmd, void *param);
    int32_t (*gpio_write) (uint32_t val, uint32_t mask);    /
    int32_t (*gpio_read) (uint32_t *val, uint32_t mask);    /
} DEV_GPIO, * DEV_GPIO_PTR;
```

Arduino Pins

IO0	Bit 0	gpio4b_2[0]
IO1	Bit 0	gpio4b_2[1]
IO2	•	gpio4b_2[2]
IO3	Bit 2	gpio4b_2[3]

1. **gpio_open** :
   Open a gpio device with pre-defined IO direction.
   param :
   uint32 dir : to set the pins as input(0) or output(1)

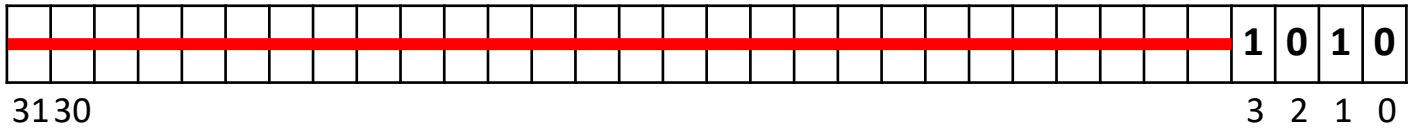
Take **gpio4b_2** for example



GPIO - API (gpio_open example)

EX :

gpio4b_2[0] : input
gpio4b_2[1] : output
gpio4b_2[2] : input
gpio4b_2[3] : output




```
DEV_GPIO_PTR gpio_4b2 = gpio_get_dev(DFSS_GPIO_4B2_ID);
gpio4b_2->gpio_open(0xa);        // Hexadecimal
//gpio4b_2->gpio_open(0b1010);   // Binary
//gpio4b_2->gpio_open(10);       // Decimal
```

But, it is poor readability ! (可讀性很差)

```
#define GPIO4B2_0_OFFSET    0
#define GPIO4B2_1_OFFSET    1
#define GPIO4B2_2_OFFSET    2
#define GPIO4B2_3_OFFSET    3
```

```
   1000
or 0010
------
   1010
```

```
DEV_GPIO_PTR gpio_4b2 = gpio_get_dev(DFSS_GPIO_4B2_ID);
gpio_4b2->gpio_open(1<<GPIO4B2_3_OFFSET | 1<<GPIO4B2_1_OFFSET);
```

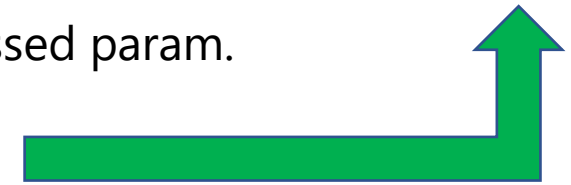
GPIO - API (gpio_close & gpio_control)

```
typedef struct dev_gpio {
    DEV_GPIO_INFO gpio_info;        /*!< gpio device information */
    int32_t (*gpio_open) (uint32_t dir);    /*!< open gpio device with pre-defined gpio direction */
    int32_t (*gpio_close) (void);       /*!< close gpio device */
    int32_t (*gpio_control) (uint32_t ctrl_cmd, void *param);   /*!< control gpio device */
    int32_t (*gpio_write) (uint32_t val, uint32_t mask);    /*!< write gpio device with val, only write the masked bits */
    int32_t (*gpio_read) (uint32_t *val, uint32_t mask);    /*!< read gpio device val, only read the masked bits */
} DEV_GPIO, * DEV_GPIO_PTR;
```

2. ***gpio_close*** :
   Close a gpio device.

3. ***gpio_control*** :
   Control an gpio device by ctrl_cmd, with passed param.
   param :
   uint32_t ctrl_cmd : control command
   void *param : parameters that maybe argument of the command, or return values of the command

```
#define GPIO_CMD_SET_BIT_DIR_INPUT          DEV_SET_SYSCMD(0)
#define GPIO_CMD_SET_BIT_DIR_OUTPUT         DEV_SET_SYSCMD(1)
#define GPIO_CMD_GET_BIT_DIR                DEV_SET_SYSCMD(2)
```



GPIO - API (gpio_control example)

```
#define GPIO_CMD_SET_BIT_DIR_INPUT          DEV_SET_SYSCMD(0)
#define GPIO_CMD_SET_BIT_DIR_OUTPUT         DEV_SET_SYSCMD(1)
#define GPIO_CMD_GET_BIT_DIR                DEV_SET_SYSCMD(2)
```

EX :

gpio4b_2[0] : input
gpio4b_2[1] : output
gpio4b_2[2] : input
gpio4b_2[3] : output

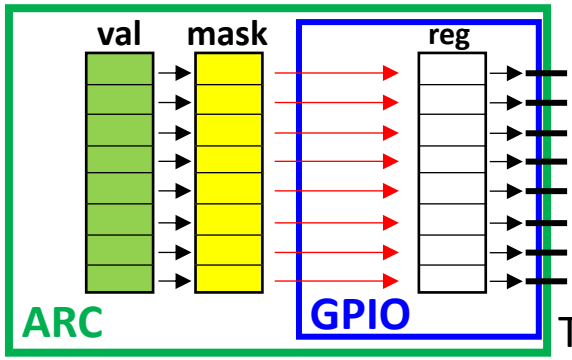
```
DEV_GPIO_PTR gpio_4b2 = gpio_get_dev(DFSS_GPIO_4B2_ID);

uint32_t setPin = 1<<GPIO4B2_3_OFFSET | 1<<GPIO4B2_1_OFFSET;
uint32_t getDir;

gpio_4b2->gpio_open(0);
gpio_4b2->gpio_control(GPIO_CMD_GET_BIT_DIR, &getDir);
EMBARC_PRINTF("\n%d\n", getDir);

gpio_4b2->gpio_control(GPIO_CMD_SET_BIT_DIR_OUTPUT, (void*)setPin);
//gpio_4b2->gpio_control(GPIO_CMD_SET_BIT_DIR_OUTPUT, (void*)(1<<GPIO4B2_3_OFFSET | 1<<GPIO4B2_1_OFFSET));
gpio_4b2->gpio_control(GPIO_CMD_GET_BIT_DIR, &getDir);
EMBARC_PRINTF("%b\n", getDir);

gpio_4b2->gpio_close();
```

```
embARC Build Time: Oct 31 2020, 21:03:12
Compiler Version: ARC GNU, 9.3.1 20200315

0
1010
```

GPIO - API (gpio_write)

```
typedef struct dev_gpio {
    DEV_GPIO_INFO gpio_info;        /*!< gpio device information */
    int32_t (*gpio_open) (uint32_t dir);    /*!< open gpio device with pre-defined gpio direction */
    int32_t (*gpio_close) (void);       /*!< close gpio device */
    int32_t (*gpio_control) (uint32_t ctrl_cmd, void *param);   /*!< control gpio device */
    int32_t (*gpio_write) (uint32_t val, uint32_t mask);    /*!< write gpio device with val, only write the masked bits */
    int32_t (*gpio_read) (uint32_t *val, uint32_t mask);    /*!< read gpio device val, only read the masked bits */
} DEV_GPIO, * DEV_GPIO_PTR;
```

4. ***gpio_write*** :
   Write gpio device with val, **only write the masked bits**
   param :
   uint32_t val : the data that need to write to gpio
   uint32_t mask : gpio bit mask

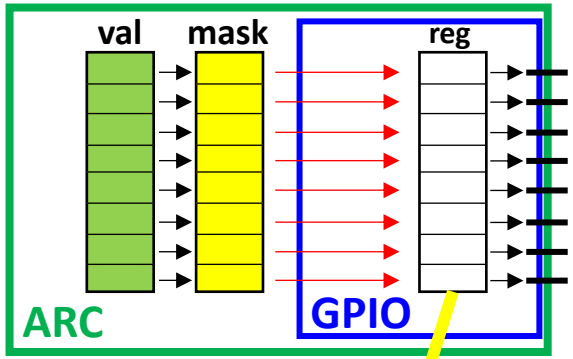


Take 8bits for example

val	0	1	0	0	1	0	1	1
mask	0	0	1	0	1	1	0	1
reg (Before Write)	0	0	0	1	0	1	1	1
reg (After Write)	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**1**

GPIO - API (gpio_write Question!)

The bit you want to maintain → mask = 0
The bit you want to change → mask = 1



Q1 : Why do we need to use mask?

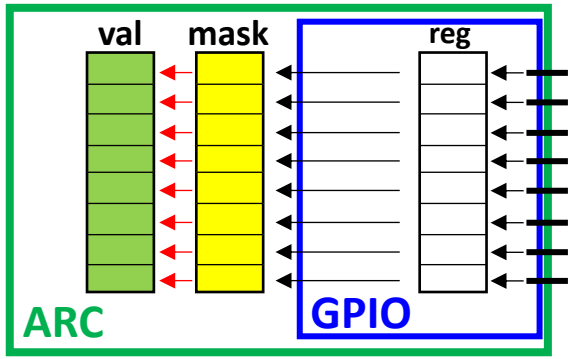
Q2 : The right side is the code of gpio_write, please explain the meaning line by line.

```
int32_t ss_gpio_write(SS_GPIO_DEV_CONTEXT *ctx, uint32_t val, uint32_t mask)
{
    uint32_t temp_val;

    io_gpio_read(ctx->dev_id, &temp_val);
    temp_val &= (~mask);
    val &= mask;
    io_gpio_write(ctx->dev_id, temp_val | val);

    return 0;
}
```

GPIO - API (gpio_read)

```
typedef struct dev_gpio {
    DEV_GPIO_INFO gpio_info;        /*!< gpio device information */
    int32_t (*gpio_open) (uint32_t dir);    /*!< open gpio device with pre-defined gpio direction */
    int32_t (*gpio_close) (void);       /*!< close gpio device */
    int32_t (*gpio_control) (uint32_t ctrl_cmd, void *param);   /*!< control gpio device */
    int32_t (*gpio_write) (uint32_t val, uint32_t mask);    /*!< write gpio device with val, only write the masked bits */
    int32_t (*gpio_read) (uint32_t *val, uint32_t mask);    /*!< read gpio device val, only read the masked bits */
} DEV_GPIO, * DEV_GPIO_PTR;
```

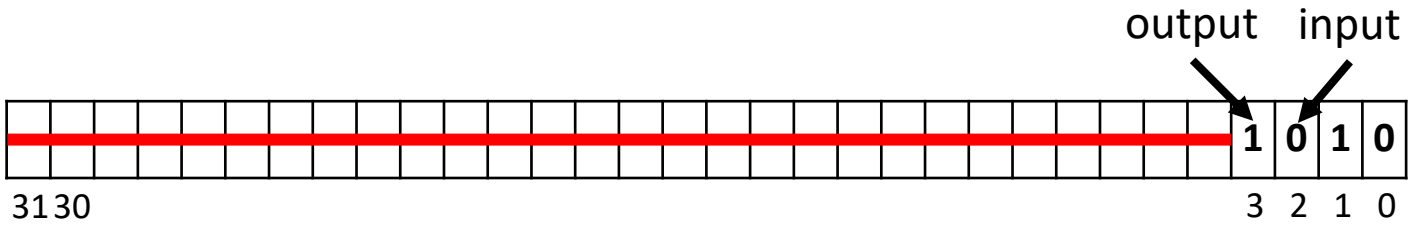
5. **gpio_read** :
   Read gpio device val, **only read the masked bits**
   param :
   uint32_t *val : save read value to *val
   uint32_t mask : gpio bit mask



Take 8bits for example

data	1	0	1	0	1	0	1	0
mask	0	1	1	0	1	1	0	1
stored to val	0	0	1	0	1	0	0	0

GPIO - Summary




```
typedef struct dev_gpio {
    DEV_GPIO_INFO gpio_info;        /*!< gpio device information */
    int32_t (*gpio_open) (uint32_t dir);    /*!< open gpio device with pre-defined gpio direction */
    int32_t (*gpio_close) (void);       /*!< close gpio device */
    int32_t (*gpio_control) (uint32_t ctrl_cmd, void *param);   /*!< control gpio device */
    int32_t (*gpio_write) (uint32_t val, uint32_t mask);    /*!< write gpio device with val, only write the masked bits */
    int32_t (*gpio_read) (uint32_t *val, uint32_t mask);    /*!< read gpio device val, only read the masked bits */
} DEV_GPIO, * DEV_GPIO_PTR;
```

Control Command :

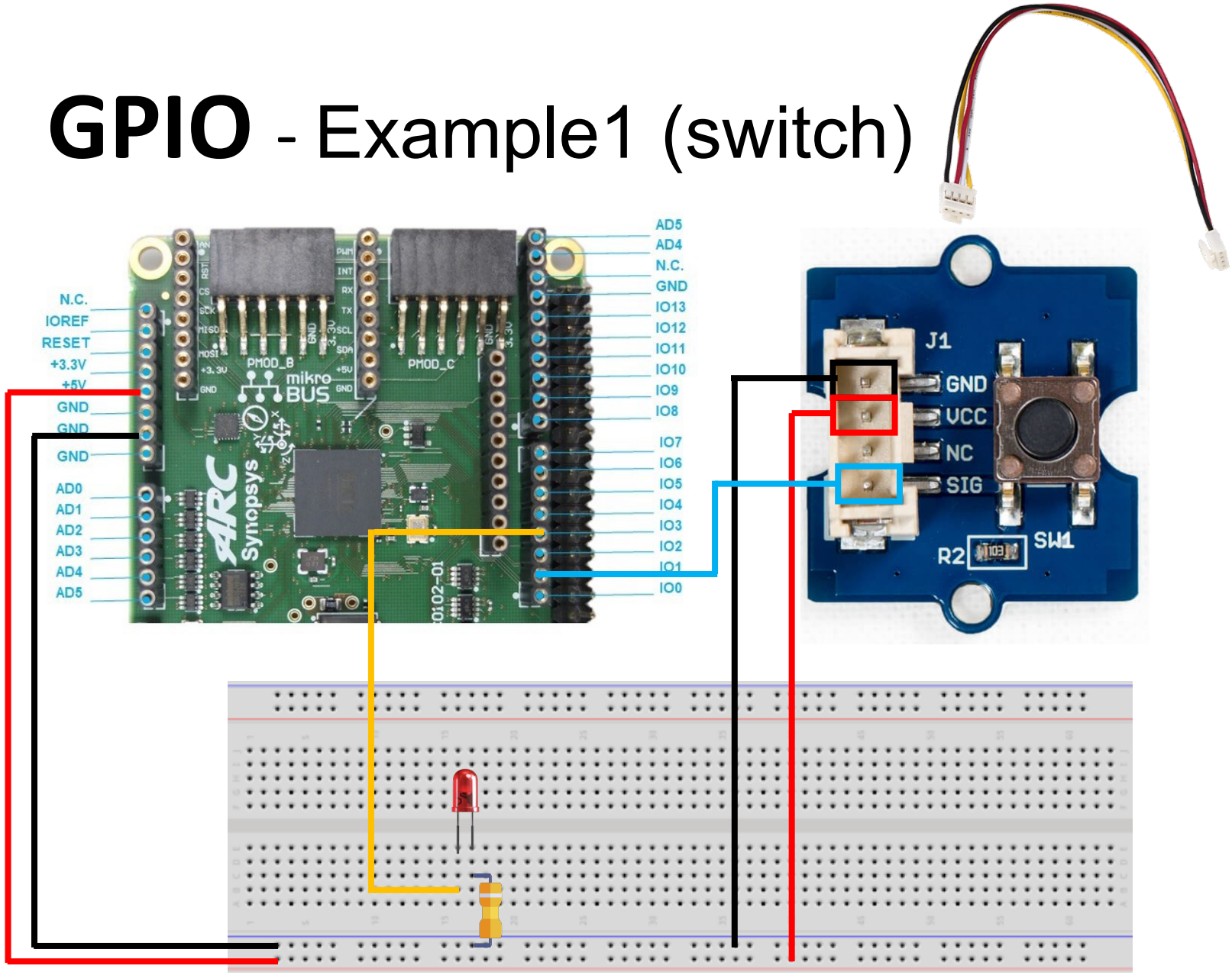
```
#define GPIO_CMD_SET_BIT_DIR_INPUT          DEV_SET_SYSCMD(0)
#define GPIO_CMD_SET_BIT_DIR_OUTPUT         DEV_SET_SYSCMD(1)
#define GPIO_CMD_GET_BIT_DIR                DEV_SET_SYSCMD(2)
```

```
#define GPIO4B2_0_OFFSET    0
#define GPIO4B2_1_OFFSET    1
#define GPIO4B2_2_OFFSET    2
#define GPIO4B2_3_OFFSET    3
```

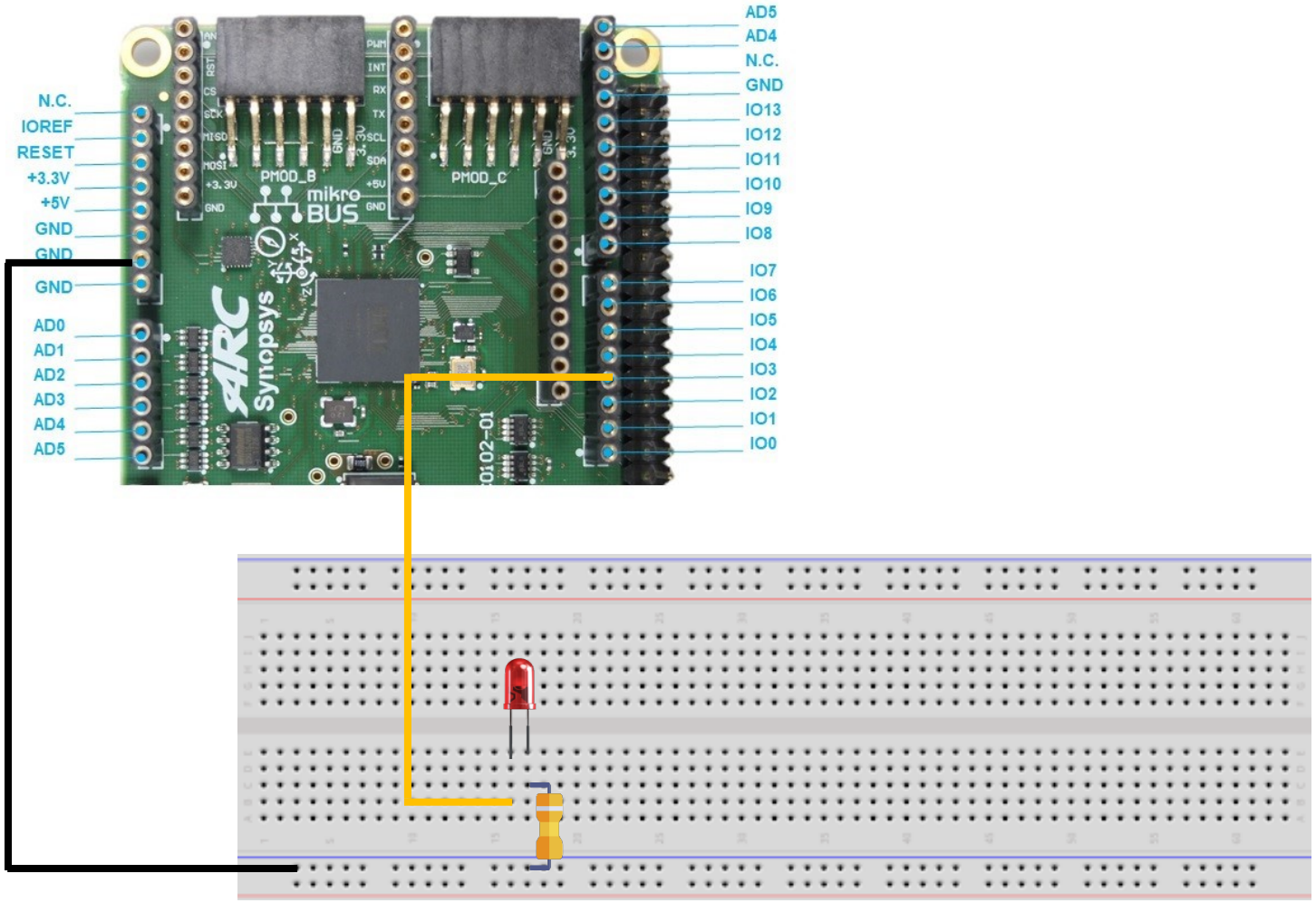
```
DEV_GPIO_PTR gpio_4b2 = gpio_get_dev(DFSS_GPIO_4B2_ID);
gpio_4b2->gpio_open(1<<GPIO4B2_3_OFFSET | 1<<GPIO4B2_1_OFFSET);
```

val	0	1	0	0	1	0	1	1
mask	0	0	1	0	1	1	0	1
reg (Before Write)	0	0	0	1	0	1	1	1
reg (After Write)	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**1**

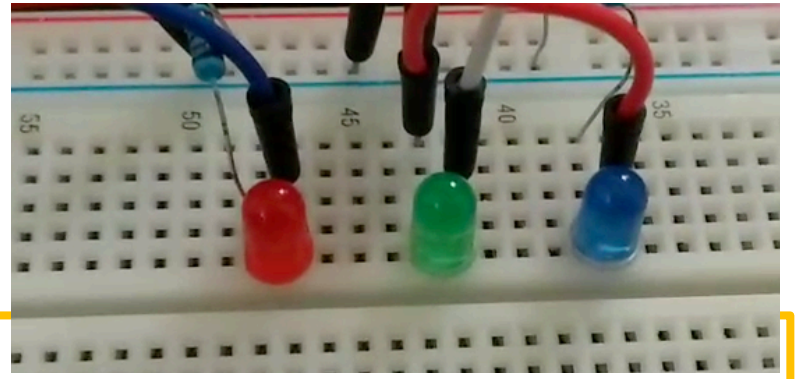
GPIO - Example1 (switch)



GPIO - Example2 (blinky)



GPIO - LAB



1. **LAB1**

   Components : Red(R), Green(G), Blue(B) LED, and button

   Function :

   We need 8 mode (0 : dark, 1 : light)

   ```
   ---------------------------------------------------------------
   | 0 : R = 0, G = 0, B = 0 (INIT)                              |
   | 1 : R = 1, G = 0, B = 0                                     |
   | 2 : R = 0, G = 1, B = 0                                     |
   | 3 : R = 0, G = 0, B = 1                                     |
   | 4 : R = 1, G = 1, B = 0                                     |
   | 5 : R = 0, G = 1, B = 1                                     |
   | 6 : R = 1, G = 0, B = 1                                     |
   | 7 : R = 1, G = 1, B = 1 (Next Return to MODE 0)             |
   ---------------------------------------------------------------
   ```

   Please use button to switch MODE from 0 to 7

   ※ Key point : How to determine press one time
   (one time means click down & up)

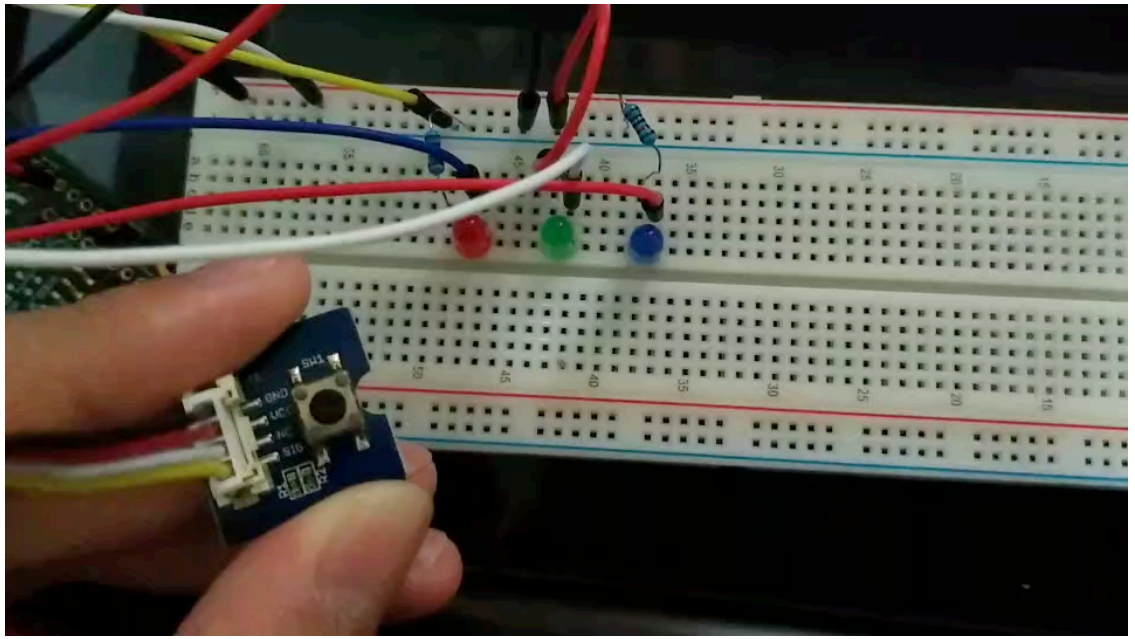
2. **LAB2**

   Components : Red(R), Green(G), Blue(B) LED

   Function :

   Please let LED blinky (light : 0.5s, dark : 0.5s)

   R : 2 secons ➔ G : 3 seconds ➔ B : 1 seconds



Can we combine LAB1 & LAB2 ?

We need 8 mode (0 : dark, 1 : light)

0 : R = 0, G = 0, B = 0 (INIT)
1 : R = 1, G = 0, B = 0
2 : R = 0, G = 1, B = 0
3 : R = 0, G = 0, B = 1
4 : R = 1, G = 1, B = 0
5 : R = 0, G = 1, B = 1
6 : R = 1, G = 0, B = 1
7 : R = 1, G = 1, B = 1 (Next Return to MODE 0)

(1) Use button to switch MODE from 0 to 7

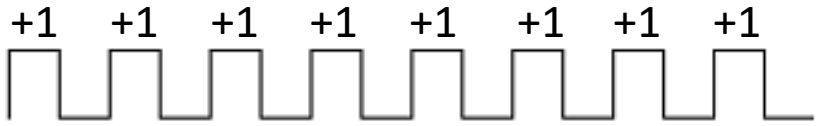
(2) Let LED blinky (light : 0.5s, dark : 0.5s)

Timer & Interrupt

Timer - Introduction

AUX_TIMERY_CTRL : Control Mode
AUX_TIMERY_LIMIT : Limit Value
AUX_TIMERY_CNT : Current Count Value

- Timer is used for counting time
- Plus one per system cycle
  (**1 sec = 144M system cycles**)



- When achieve the limit value, then return to 0 and continue counting
- There are some control modes we can choose (will introduce later)

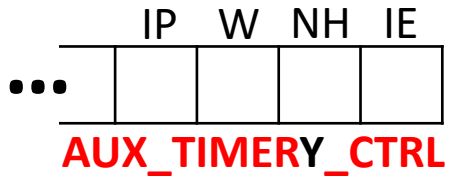
ARC IoTDK has three timers, Timer0 & Timer1 & Timer_RTC

```
#define TIMER_0     0
#define TIMER_1     1
#define TIMER_RTC   2
```

We use these two timers

Timer - Introduction

AUX_TIMERY_CTRL : Control Mode
AUX_TIMERY_LIMIT : Limit Value
AUX_TIMERY_CNT : Current Count Value



mode :

```
#define TIMER_CTRL_IE       (1 << 0)    /*!< Interrupt when count reaches limit */
#define TIMER_CTRL_NH       (1 << 1)    /*!< Count only when CPU NOT halted */
#define TIMER_CTRL_W        (1 << 2)    /*!< watchdog enable */
#define TIMER_CTRL_IP       (1 << 3)    /*!< interrupt pending */
```

IP ： a status, represents having an unresolved interrupt

ISR (中斷處理程序)
Interrupt Service Routines

Timer - API (timer_present)

```
int32_t timer_present(const uint32_t no);
int32_t timer_start(const uint32_t no, const uint32_t mode, const uint32_t val);
int32_t timer_stop(const uint32_t no);
int32_t timer_current(const uint32_t no, void* val);
int32_t timer_int_clear(const uint32_t no);
```

1. **timer_present** :
   Check whether the specific timer present
   param :
   uint32_t no : timer ID
   return :
   1 present, 0 not present

timer ID :

```
#define TIMER_0     0
#define TIMER_1     1
```

To prevent using nonexistent timer, we must use this function to check before operating the timer

```
if (timer_present(TIMER_0))
```

If (1) ➔ timer exists, then we can operate it
If (0) ➔ timer doesn't exist, we can't use it

Timer - API (timer_start)

```
#define TIMER_0      0
#define TIMER_1      1
```

```
int32_t timer_present(const uint32_t no);
int32_t timer_start(const uint32_t no, const uint32_t mode, const uint32_t val);
int32_t timer_stop(const uint32_t no);
int32_t timer_current(const uint32_t no, void* val);
int32_t timer_int_clear(const uint32_t no);
```

2. **timer_start** :
   Reset & Start the specific timer
   param :
   - uint32_t no : timer ID
   - uint32_t mode : timer mode
   - uint32_t val : timer limit value

val :

n * BOARD_CPU_CLOCK

➔ n seconds

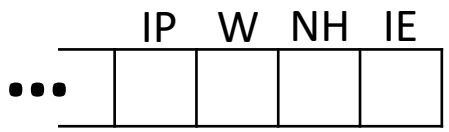
Board CPU clocks per seconds

```
#define BOARD_CPU_CLOCK          CLK_CPU

#define CLK_CPU      (BOARD_DFSS_CORE_CLK)

#define BOARD_DFSS_CORE_CLK (144000000U)
```

144 MHz

EX :

```
timer_start(TIMER_1, TIMER_CTRL_IE, 2*BOARD_CPU_CLOCK);
timer_start(TIMER_0, 0, 10*BOARD_CPU_CLOCK);
timer_start(TIMER_0, TIMER_CTRL_IE|TIMER_CTRL_IP, BOARD_CPU_CLOCK);
```



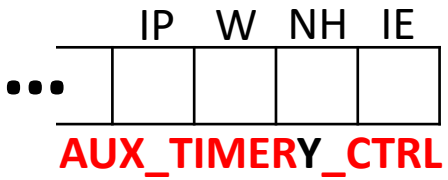
mode : AUX_TIMERY_CTRL

```
#define TIMER_CTRL_IE       (1 << 0)    /*!< Interrupt when count reaches limit */
#define TIMER_CTRL_NH       (1 << 1)    /*!< Count only when CPU NOT halted */
#define TIMER_CTRL_W        (1 << 2)    /*!< watchdog enable */
#define TIMER_CTRL_IP       (1 << 3)    /*!< interrupt pending */
```

ISR (中斷處理程序)
Interrupt Service Routines

Timer - API (timer_start code)

AUX_TIMERY_CTRL : Control Mode
AUX_TIMERY_LIMIT : Limit Value
AUX_TIMERY_CNT : Current Count Value



mode :

```
#define TIMER_CTRL_IE       (1 << 0)
#define TIMER_CTRL_NH       (1 << 1)
#define TIMER_CTRL_W        (1 << 2)
#define TIMER_CTRL_IP       (1 << 3)
```

```
int32_t timer_start(const uint32_t no, const uint32_t mode, const uint32_t val)
{
    switch (no) {
        case TIMER_0:
            _arc_aux_write(AUX_TIMER0_CTRL, 0);          Clear Mode
            _arc_aux_write(AUX_TIMER0_LIMIT, val);       Set Limit Value
            _arc_aux_write(AUX_TIMER0_CTRL, mode);       Set Mode
            _arc_aux_write(AUX_TIMER0_CNT, 0);           Clear Count Value
            break;
        case TIMER_1:
            _arc_aux_write(AUX_TIMER1_CTRL, 0);
            _arc_aux_write(AUX_TIMER1_LIMIT, val);
            _arc_aux_write(AUX_TIMER1_CTRL, mode);
            _arc_aux_write(AUX_TIMER1_CNT, 0);
            break;
        case TIMER_RTC:
            _arc_aux_write(AUX_RTC_CTRL, mode);
            break;
        default:
            return -1;
    }

    return 0;
}
```

Timer - API (timer_stop)

```
int32_t timer_present(const uint32_t no);
int32_t timer_start(const uint32_t no, const uint32_t mode, const uint32_t val);
int32_t timer_stop(const uint32_t no);
int32_t timer_current(const uint32_t no, void* val);
int32_t timer_int_clear(const uint32_t no);
```

3. **timer_stop** :
   Stop and clear the specific timer
   param :
   uint32_t no : timer ID

	IP	W	NH	IE
•••	0	0	0	0

AUX_TIMERY_CTRL

AUX_TIMERY_CTRL : Control Mode
AUX_TIMERY_LIMIT : Limit Value
AUX_TIMERY_CNT : Current Count Value

EX :

```
timer_stop(TIMER_0);
```

```
int32_t timer_stop(const uint32_t no)
{
    switch (no) {
        case TIMER_0 :
            _arc_aux_write(AUX_TIMER0_CTRL, 0);
            _arc_aux_write(AUX_TIMER0_LIMIT,0);
            _arc_aux_write(AUX_TIMER0_CNT, 0);
            break;
        case TIMER_1:
            _arc_aux_write(AUX_TIMER1_CTRL, 0);
            _arc_aux_write(AUX_TIMER1_LIMIT,0);
            _arc_aux_write(AUX_TIMER1_CNT, 0);
            break;
        case TIMER_RTC:
            _arc_aux_write(AUX_RTC_CTRL, TIMER_RTC_CLEAR);
            break;
        default:
            return -1;
    }

    return 0;
}
```

Timer - API (timer_current)

```
int32_t timer_present(const uint32_t no);
int32_t timer_start(const uint32_t no, const uint32_t mode, const uint32_t val);
int32_t timer_stop(const uint32_t no);
int32_t timer_current(const uint32_t no, void* val);
int32_t timer_int_clear(const uint32_t no);
```

4. **timer_current** :
   Get timer current count:
   uint32_t no : timer ID
   void* val : save cuurent time to *val

AUX_TIMERY_CTRL : Control Mode
AUX_TIMERY_LIMIT : Limit Value
AUX_TIMERY_CNT : Current Count Value

EX :

```
uint32_t val;
timer_current(TIMER_0, &val);
```

```
int32_t timer_current(const uint32_t no, void *val)
{
    switch (no) {
        case TIMER_0 :
            *((uint32_t *)val) = _arc_aux_read(AUX_TIMER0_CNT);
            break;
        case TIMER_1 :
            *((uint32_t *)val) = _arc_aux_read(AUX_TIMER1_CNT);
            break;
        case TIMER_RTC:
            *((uint64_t *)val) = _arc_aux_read(AUX_RTC_LOW);
            break;
        default :
            return -1;
    }

    return 0;
}
```

Interrupt - Introduction

廣義的中斷(Generalized Interrupt) :

指程序執行過程中，遇到其他需處理的事情時，暫時中止CPU上現行的程序，轉去執行相應的事件處理程序，處理完後再返回原程序被中斷處或由中斷處理程序指定的位置

狹義的中斷(Narrowly Interrupt) :

由CPU之外引發中斷事件，與當前運行的指令無關的中斷事件。
ex : I/O中斷、計時器中斷(timer)、外部信號中斷 ... etc

IRQ (中斷請求線)
Interrupt Request Line

異常(Exception) :

由CPU內部產生的中斷事件，由當前運行指令引起的中斷事件。
ex : (指令/記憶體)地址異常、算術異常(除以0)、user program需要OS提供Service時發出 ... etc

廣義的中斷 = 狹義的中斷 + 異常
Generalized Interrupt = Narrowly Interrupt + Exception

Interrupt - Introduction

中斷向量表 (Interrupt Vector Table)

When interrupt request is delivered /exception happen

1. Find Entrance (Interrupt Vector Base + Interrupt Vector)
2. Check whether the interrupt is enabled
3. If yes, go to Handler(ISR) to do something

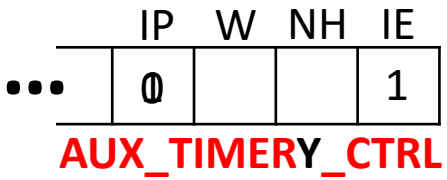
ISR (中斷處理程序)
Interrupt Service Routines

Interrupt Vector (中斷向量值)	Description	Enable (0 / 1)	Handler (ISR)
0	Reset		
1	Memory Error		
2	Instruction Error		
3	EV_MachineCheck		
4	EV_TLBMissI		
5	EV_TLBMissD		
6	EV_ProtV		
7	EV_PrivilegeV		
8	EV_SWI		
9	EV_Trap		
10	EV_Extension		
11	EV_DivZero		
12	EV_DCError		
13	EV_Malligned		
14	Unused		
15	Unused		
16	Timer0/ext int		
17	Timer1/ext int		
18	ext int		
...	...		
253	ext int		
254	ext int		
255	ext int		

exceptions

external interrupt

Interrupt - Introduction



IRQ (中斷請求線)
Interrupt Request Line

Timer interrupt

1. IE = 1, timer start
2. When reached limit ➔ IP = 1
   CNT return to 0, and continue updating
3. Interrupt Vector Base + Interrupt Vector
4. Check whether the interrupt is enabled
5. If No, do nothing
   If yes, go to Handler(ISR) to do something
   ISR : IP → 0, do something, then return

AUX_TIMERY_CTRL : Control Mode
AUX_TIMERY_LIMIT : Limit Value
AUX_TIMERY_CNT : Current Count Value

Interrupt Vector (中斷向量值)	Description	Enable (0 / 1)	Handler (ISR)
0	Reset		
1	Memory Error		
2	Instruction Error		
3	EV_MachineCheck		
4	EV_TLBMissI		
5	EV_TLBMissD		
6	EV_ProtV		
7	EV_PrivilegeV		
8	EV_SWI		
9	EV_Trap		
10	EV_Extension		
11	EV_DivZero		
12	EV_DCError		
13	EV_Malligned		
14	Unused		
15	Unused		
16	Timer0/ext int		
17	Timer1/ext int		
18	ext int		
...	...		
253	ext int		
254	ext int		
255	ext int		

exceptions

external interrupt

Interrupt - API

```
#define INTNO_TIMER0 16
#define INTNO_TIMER1 17
```

```
int32_t int_disable(const uint32_t intno);
int32_t int_enable(const uint32_t intno);
int32_t int_handler_install(const uint32_t intno, INT_HANDLER handler);
int32_t int_pri_get(const uint32_t intno);
int32_t int_pri_set(const uint32_t intno, int32_t intpri);
```

1. **int_disable** :
   Disable the specific interrupt
   param :
   uint32_t intno : specific interrupt number
2. **int_enable** :
   Enable the specific interrupt
   param :
   uint32_t intno : specific interrupt number
3. **int_handler_install** :
   Install an interrupt handler
   param :
   uint32_t intno : specific interrupt number
   INT_HANDLER handler : interrupt handler function

```
typedef void (*INT_HANDLER) (void *ptr);
```

```
void t1_isr(void *ptr)
{
    // ...
}
```

ISR (中斷處理程序)
Interrupt Service Routines

EX :

```
int_disable(INTNO_TIMER1);
```

```
int_enable(INTNO_TIMER0);
```

```
int_handler_install(INTNO_TIMER1, t1_isr);
```

Interrupt - API

```
#define INTNO_TIMER0 16
#define INTNO_TIMER1 17
```

```
int32_t int_disable(const uint32_t intno);
int32_t int_enable(const uint32_t intno);
int32_t int_handler_install(const uint32_t intno, INT_HANDLER handler);
int32_t int_pri_get(const uint32_t intno);
int32_t int_pri_set(const uint32_t intno, int32_t intpri);
```

4. **int_pri_get** :
   Get the priority of the specific interrupt
   param :
   uint32_t intno : specific interrupt number
   return :
   the priority of the specific interrupt
5. **int_pri_set** :
   Set the priority of the specific interrupt
   param :
   uint32_t intno : specific interrupt number
   int32_t intpri : -4(the highest) ~ -1(the lowest)

INT_PRI_MIN = -4
INT_PRI_MAX = -1

Initial : everyone is -1
The higher priority can interrupt the lower
The same or less can't interrupt
> : can interrupt
<= : can't

EX :

```
int pri = 0;
pri = int_pri_get(INTNO_TIMER0);
```

```
int_pri_set(INTNO_TIMER0, INT_PRI_MIN);
```

```
int_pri_set(INTNO_TIMER1, -2);
```

Timer - API (timer_int_clear)

	IP	W	NH	IE
...	1	X	X	X

AUX_TIMERY_CTRL

```
int32_t timer_present(const uint32_t no);
int32_t timer_start(const uint32_t no, const uint32_t mode, const uint32_t val);
int32_t timer_stop(const uint32_t no);
int32_t timer_current(const uint32_t no, void* val);
◎ int32_t timer_int_clear(const uint32_t no);
```

```
timer_start(TIMER_1, TIMER_CTRL_IP, 5*BOARD_CPU_CLOCK);
```

5. timer_int_clear :
   Clear the interrupt pending bit of the timer
   uint32_t no : timer ID

```
    1XXX
&   0111
-------
    0XXX
```

mode :

```
#define TIMER_CTRL_IE       (1 << 0)
#define TIMER_CTRL_NH       (1 << 1)
#define TIMER_CTRL_W        (1 << 2)
#define TIMER_CTRL_IP       (1 << 3)
```

EX :

ISR (中斷處理程序)
Interrupt Service Routines

```
void t1_isr(void *ptr)
{
    timer_int_clear(TIMER_1);
    EMBARC_PRINTF("hello\n");
}
```

AUX_TIMERY_CTRL : Control Mode
AUX_TIMERY_LIMIT : Limit Value
AUX_TIMERY_CNT : Current Count Value

```
int32_t timer_int_clear(const uint32_t no)
{
    uint32_t val;

    switch (no) {
        case TIMER_0 :
            val = _arc_aux_read(AUX_TIMER0_CTRL);
            val &= ~TIMER_CTRL_IP;
            _arc_aux_write(AUX_TIMER0_CTRL, val);
            break;
        case TIMER_1 :
            val = _arc_aux_read(AUX_TIMER1_CTRL);
            val &= ~TIMER_CTRL_IP;
            _arc_aux_write(AUX_TIMER1_CTRL, val);
            break;
        default :
            return -1;
    }

    return 0;
}
```

Timer - Board Timer

```
#define BOARD_SYS_TIMER_ID          TIMER_0
#define BOARD_SYS_TIMER_INTNO           INTNO_TIMER0
#define BOARD_SYS_TIMER_HZ          (1000)
```

```
static uint32_t cyc_hz_count = (BOARD_CPU_CLOCK / BOARD_SYS_TIMER_HZ);
```

```
void board_init(void)
{
    /* cpu to CLK_CPU */
    pll_fout_config(CLK_CPU/1000000);
    timer_init();
    iotdk_mux_init();
    iotdk_timer_init();
}
```

```
void iotdk_timer_init(void)
{
    if (timer_present(BOARD_SYS_TIMER_ID)) {
        int_disable(BOARD_SYS_TIMER_INTNO); /* disable first then enable */
        int_handler_install(BOARD_SYS_TIMER_INTNO, iotdk_timer_isr);
        timer_start(BOARD_SYS_TIMER_ID, TIMER_CTRL_IE|TIMER_CTRL_NH, cyc_hz_count);

        int_enable(BOARD_SYS_TIMER_INTNO);
    }
}
```

```
void timer_init(void)
{
    if (timer_present(TIMER_0)) {
        timer_stop(TIMER_0);
    }

    if (timer_present(TIMER_1)) {
        timer_stop(TIMER_1);
    }

    if (timer_present(TIMER_RTC)) {
        timer_stop(TIMER_RTC);
    }
}
```

```
static void iotdk_timer_isr(void *ptr)
{
    timer_int_clear(BOARD_SYS_TIMER_ID);

    board_timer_update(BOARD_SYS_TIMER_HZ);
}
```

Timer - Board Timer

```c
void board_delay_ms(uint32_t ms, uint8_t os_compat)
{
    uint64_t start_us, us_delayed;

#ifdef ENABLE_OS
    if (os_compat == OSP_DELAY_OS_COMPAT_ENABLE) {
        /** \todo add different os delay functions */
#ifdef OS_FREERTOS
        vTaskDelay(ms);
        return;
#endif
    }
#endif
    us_delayed = ((uint64_t)ms * 1000);
    start_us = board_get_cur_us();
    while ((board_get_cur_us() - start_us) < us_delayed);
}
```

```c
uint64_t board_get_cur_us(void)
{
    uint32_t sub_us;
    uint64_t total_us;
    timer_current(TIMER_0, &sub_us);

    sub_us = ((uint64_t)sub_us * 1000000) / BOARD_CPU_CLOCK;
    total_us = ((uint64_t)OSP_GET_CUR_MS()) * 1000 + (uint64_t)sub_us;

    return total_us;
}
```

If you want to use board_delay_ms ➔ Don't modify TIMER0
You can try to reset TIMER0, and use board_delay_ms to check

Timer - Summary

```
#define TIMER_0     0
#define TIMER_1     1
```

```
val :
        n   * BOARD_CPU_CLOCK
   ➔    n   seconds
```

```
int32_t timer_present(const uint32_t no);
int32_t timer_start(const uint32_t no, const uint32_t mode, const uint32_t val);
int32_t timer_stop(const uint32_t no);
int32_t timer_current(const uint32_t no, void* val);
int32_t timer_int_clear(const uint32_t no);
```

mode :

```
#define TIMER_CTRL_IE       (1 << 0)
#define TIMER_CTRL_NH       (1 << 1)
#define TIMER_CTRL_W        (1 << 2)
#define TIMER_CTRL_IP       (1 << 3)
```

Interrupt - Summary

```
int32_t int_disable(const uint32_t intno);
int32_t int_enable(const uint32_t intno);
int32_t int_handler_install(const uint32_t intno, INT_HANDLER handler);
int32_t int_pri_get(const uint32_t intno);
int32_t int_pri_set(const uint32_t intno, int32_t intpri);
```

```
#define INTNO_TIMER0 16
#define INTNO_TIMER1 17
```

INT_PRI_MIN = -4
INT_PRI_MAX = -1

```
void t1_isr(void *ptr)
{
    // Clear IP first
    timer_int_clear(TIMER_1);

    // ...
}
```

1. Always use timer_present to check before use it
2. Use timer_stop and int_disable before reset timer
3. Enable interrupt after installing a handler(ISR)
4. After all set is fine ➔ time_start

```
if(timer_present(TIMER_1))
{
    timer_stop(TIMER_1);
    int_disable(INTNO_TIMER1);
    int_handler_install(INTNO_TIMER1, t1_isr);
    int_enable(INTNO_TIMER1);
    timer_start(TIMER_1, TIMER_CTRL_IE|TIMER_CTRL_IP, 5*BOARD_CPU_CLOCK);
}
```

Timer - Example1 (observation)

Components : None

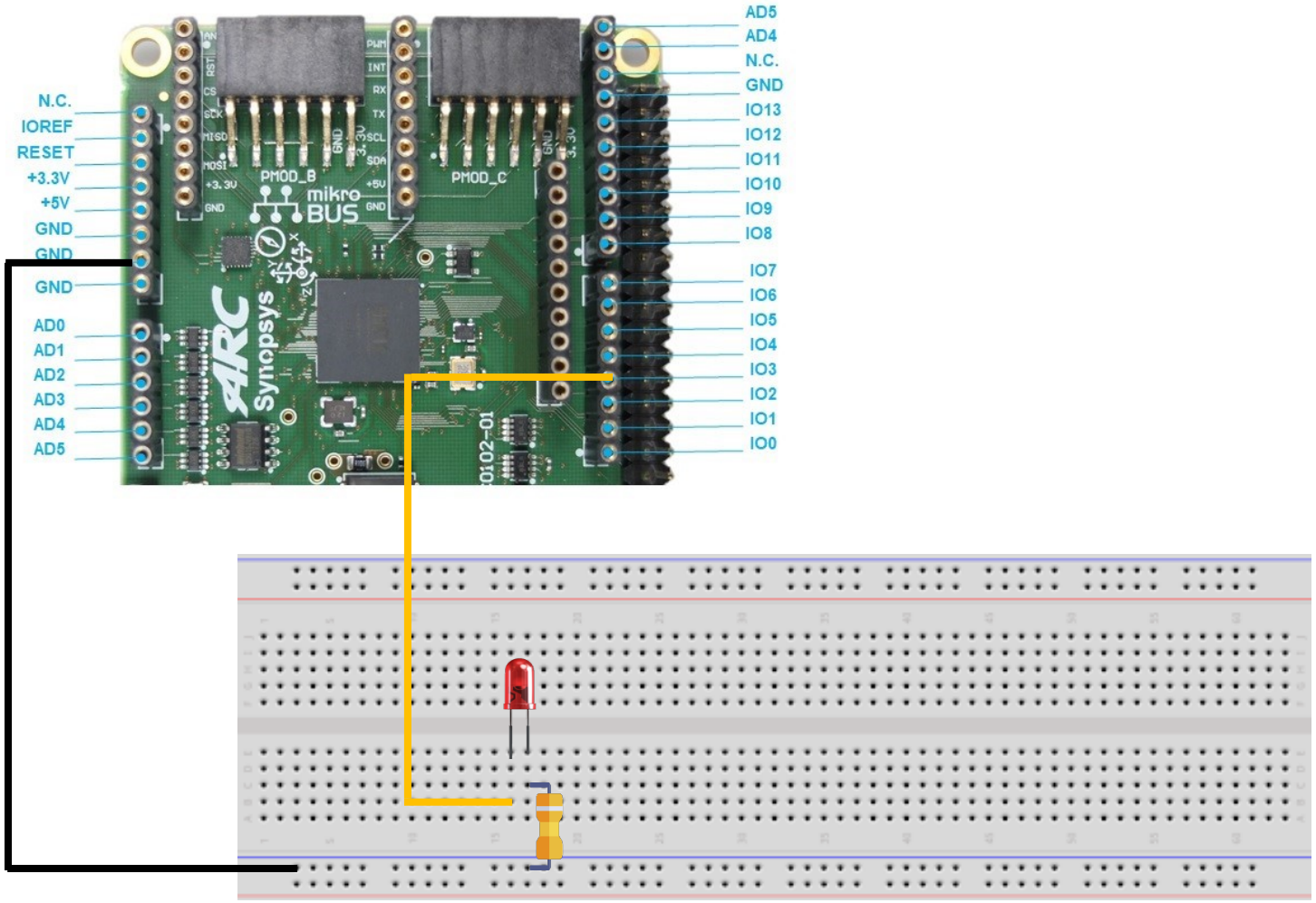
Just run the code, and check out the printed in the window

1. **Run the Example1 code, and check out**

Q : 2. **Try to comment line#98, and uncomment line#99, then run & check out**

Q : 3. **Try to comment line#70、#71, then run & check out**

Timer - Example2 (blinky)



TIMER & INTERRUPT - LAB

3. LAB3

Components : Red(R), Green(G), Blue(B) LED, and button

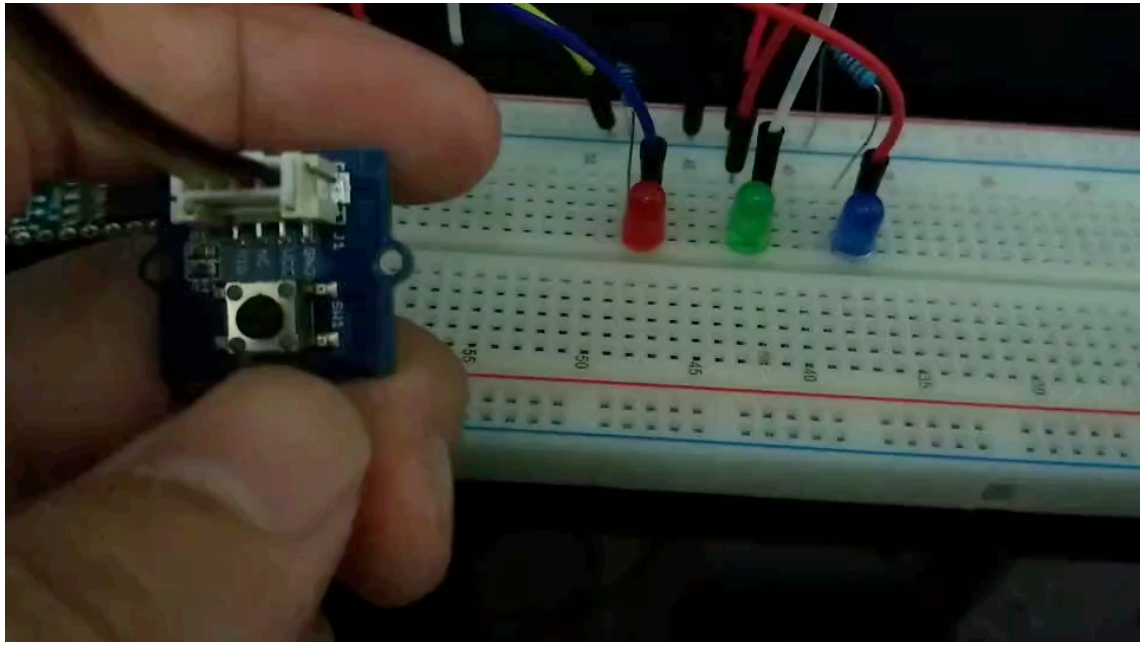
Function :

We need 8 mode (0 : dark, 1 : light)

0 : R = 0, G = 0, B = 0 (INIT)
1 : R = 1, G = 0, B = 0
2 : R = 0, G = 1, B = 0
3 : R = 0, G = 0, B = 1
4 : R = 1, G = 1, B = 0
5 : R = 0, G = 1, B = 1
6 : R = 1, G = 0, B = 1
7 : R = 1, G = 1, B = 1 (Next Return to MODE 0)

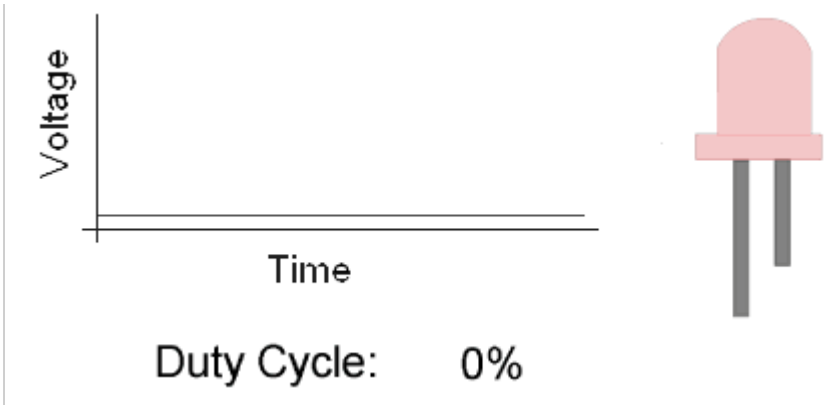
(1) Use button to switch MODE from 0 to 7

(2) Let LED blinky (light : 0.5s, dark : 0.5s)



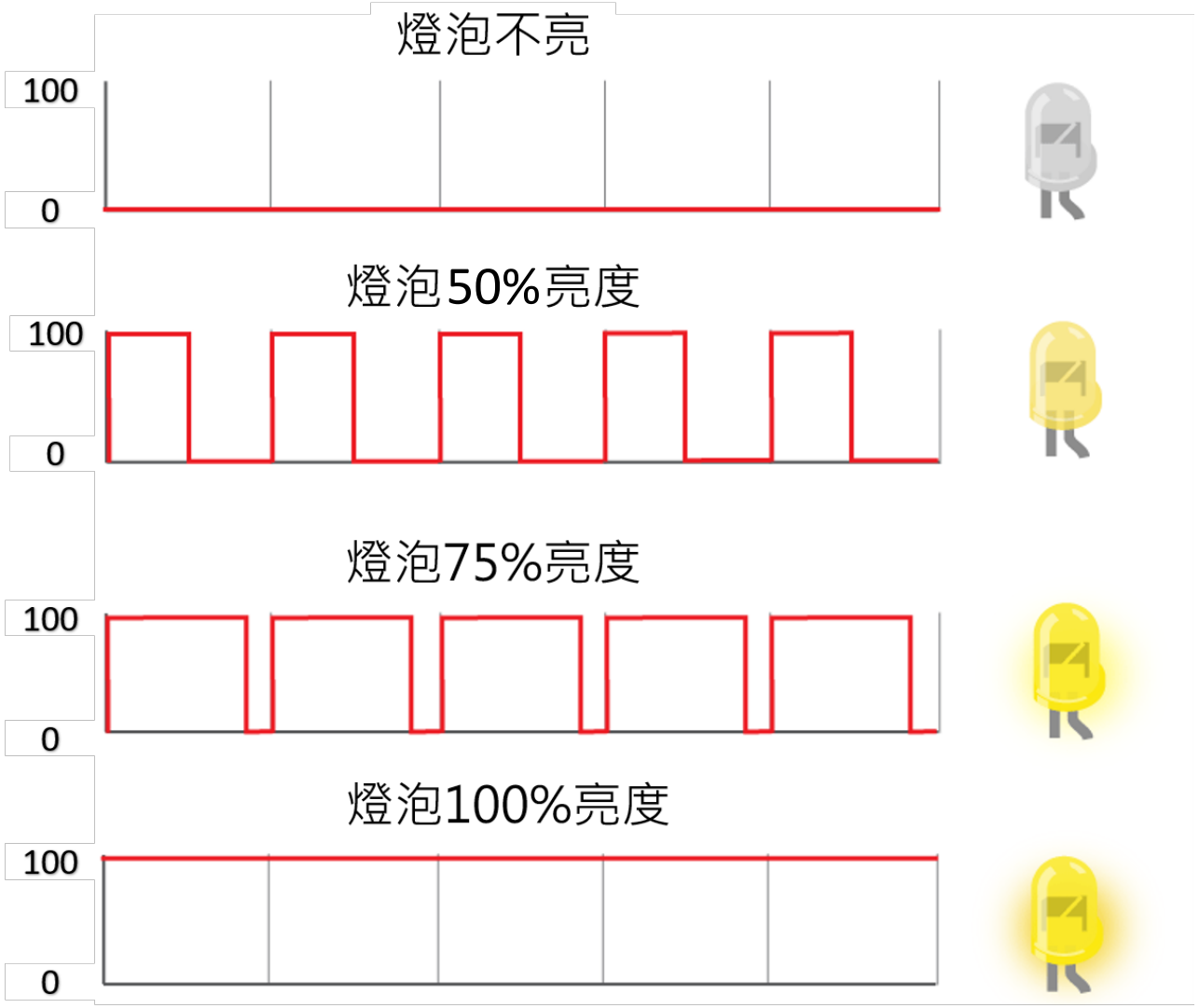
PWM

PWM - Introduction



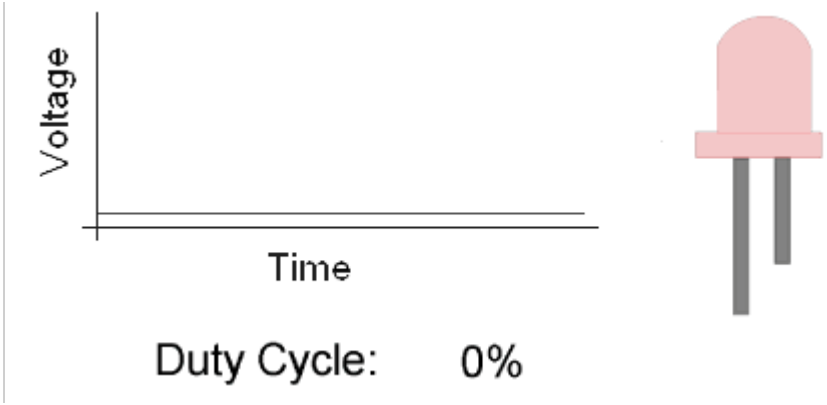
1. **Frequency :**
   n*cycles per second (Hz)
   too small => we only see long 1 & long 0 changing
   too big => the board can't handle
   Typically, we choose 100000Hz (100kHz)

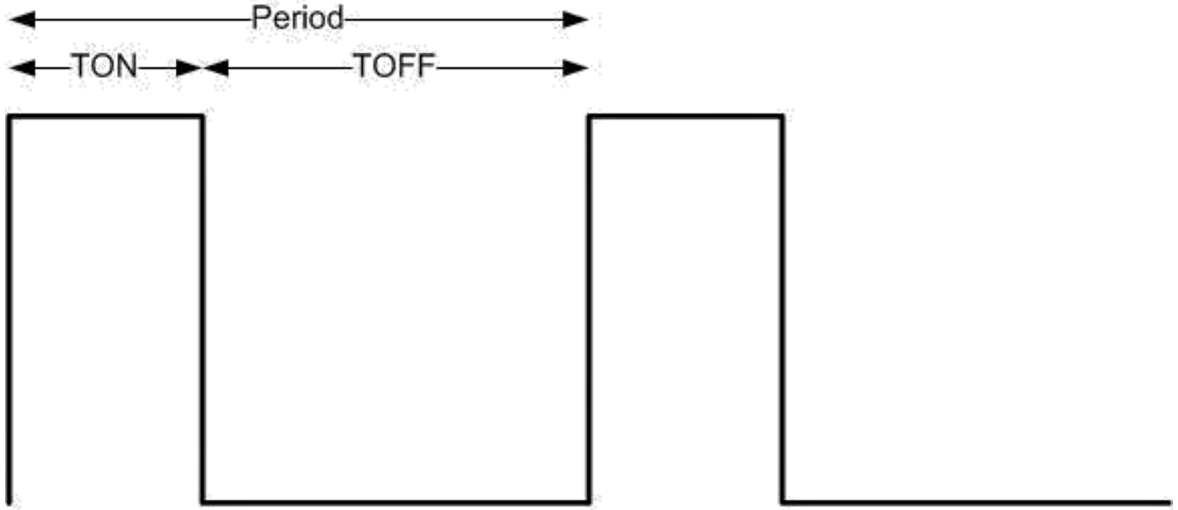
2. **Duty Cycle :**
   percentage of high voltage per Cycle



燈泡的實際亮度只有0%跟100%，但是透過閃爍，讓你誤以為他亮度減弱

PWM - Introduction





1. **Frequency :**
   n*cycles per second (Hz)
   too small => we only see long 1 & long 0 changing
   too big => the board can't handle
   Typically, we choose 100000Hz

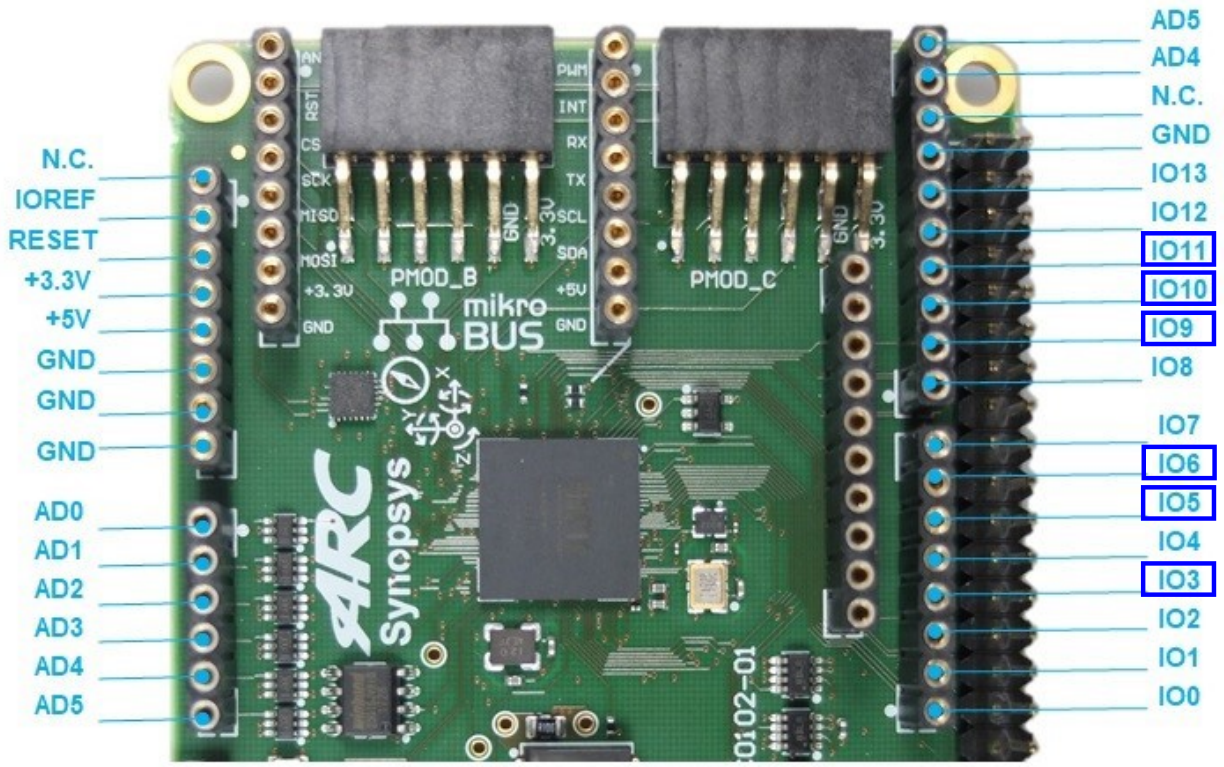
2. **Duty Cycle :**
   percentage of high voltage per Cycle

Period = TON + TOFF

Frequency = 1 / Period

$$\textbf{Duty Cycle} = \frac{TON}{TON + TOFF} * 100$$

PWM - Arduino Pins



IO3	Bit 2	gpio4b_2[3]	•	pwm0
IO4	•	gpio8b_2[0]	•	•
IO5	Bit 3	gpio8b_2[1]	•	pwm1
IO6	Bit 4	gpio8b_2[2]	•	pwm2
IO7	•	gpio8b_2[3]	•	•
IO8	•	gpio8b_2[4]	•	•
IO9	Bit 5	gpio8b_2[5]	•	pwm3
IO10	Bit 1/6	gpio8b_2[6]	spi2_cs_n	pwm4
IO11	Bit 1/7	gpio8b_2[7]	spi2_mosi	pwm5

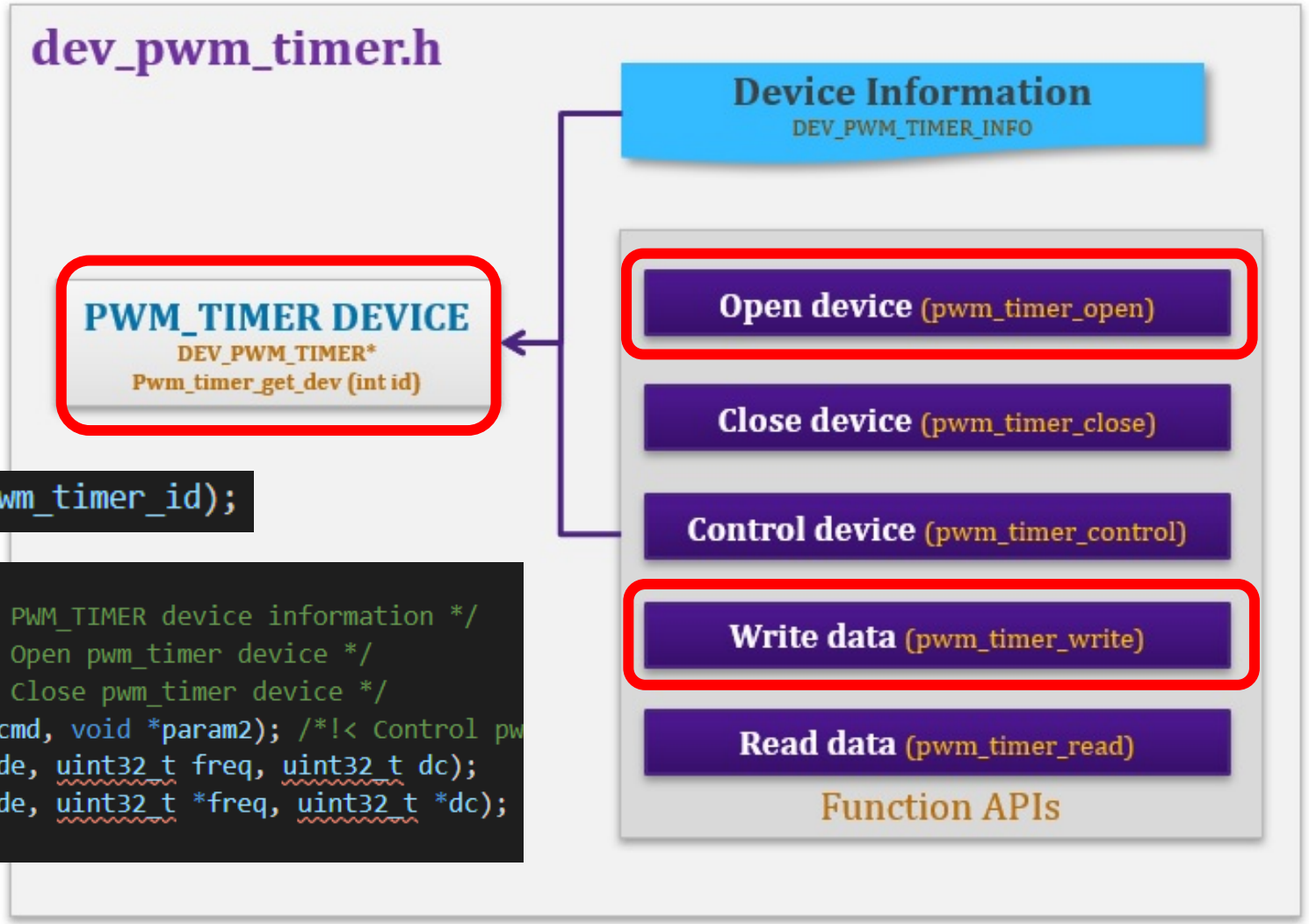
PWM - Change Pins setting

GPIO -> PWM

```c
#include "embARC.h"
#include "embARC_debug.h"

void arduino_pin_init(void)
{
    io_arduino_config(ARDUINO_PIN_3, ARDUINO_PWM, IO_PINMUX_ENABLE);//pwm timer ch0
    io_arduino_config(ARDUINO_PIN_5, ARDUINO_PWM, IO_PINMUX_ENABLE);//pwm timer ch1
    io_arduino_config(ARDUINO_PIN_6, ARDUINO_PWM, IO_PINMUX_ENABLE);//pwm timer ch2
    io_arduino_config(ARDUINO_PIN_9, ARDUINO_PWM, IO_PINMUX_ENABLE);//pwm timer ch3
    io_arduino_config(ARDUINO_PIN_10, ARDUINO_PWM, IO_PINMUX_ENABLE);//pwm timer ch4
    io_arduino_config(ARDUINO_PIN_11, ARDUINO_PWM, IO_PINMUX_ENABLE);//pwm timer ch5
}

int main(void)
{
    /* change pins settings from gpio to pwm */
    arduino_pin_init();

    // ...

    return E_SYS
}
```

PWM - API

Link :
https://embarc.org/embarc_osp/doc/build/html/device/pwm_timer.html




```
DEV_PWM_TIMER_PTR pwm_timer_get_dev(int32_t pwm_timer_id);
```

```
typedef struct dev_pwm_timer {
    DEV_PWM_TIMER_INFO pwm_timer_info;              /*!< PWM_TIMER device information */
    int32_t (*pwm_timer_open) (void);               /*!< Open pwm_timer device */
    int32_t (*pwm_timer_close) (void);              /*!< Close pwm_timer device */
    int32_t (*pwm_timer_control) (uint32_t ch, uint32_t cmd, void *param2); /*!< Control pw
    int32_t (*pwm_timer_write) (uint32_t ch, uint32_t mode, uint32_t freq, uint32_t dc);
    int32_t (*pwm_timer_read) (uint32_t ch, uint32_t *mode, uint32_t *freq, uint32_t *dc);
} DEV_PWM_TIMER, * DEV_PWM_TIMER_PTR;
```

PWM - API (pwm_timer_get_dev)

```
DEV_PWM_TIMER_PTR pwm_timer_get_dev(int32_t pwm_timer_id);
```

```
#define DW_PWM_TIMER_0_ID                0

#define DW_PWM_TIMER_0_CH_MAX_COUNT      6
```

EX :

```
DEV_PWM_TIMER_PTR pwm_timer_0 = pwm_timer_get_dev(DW_PWM_TIMER_0_ID);
```

```
DEV_PWM_TIMER_PTR pwm_timer_get_dev(int32_t pwm_timer_id)
{
    static uint32_t install_flag = 0;

    /* intall device objects */
    if (install_flag == 0) {
        install_flag = 1;
        dw_pwm_timer_all_install();
    }

    switch (pwm_timer_id) {
#if (USE_DW_PWM_TIMER_0)

        case DW_PWM_TIMER_0_ID:
            return &dw_pwm_timer_0;
            break;
#endif
        default:
            break;
    }

    return NULL;
}
```

```
typedef struct dev_pwm_timer {
    DEV_PWM_TIMER_INFO pwm_timer_info;                /*!< PWM_TIMER device information */
    int32_t (*pwm_timer_open) (void);                 /*!< Open pwm_timer device */
    int32_t (*pwm_timer_close) (void);                /*!< Close pwm_timer device */
    int32_t (*pwm_timer_control) (uint32_t ch, uint32_t cmd, void *param2); /*!< Control pwm_timer device */
    int32_t (*pwm_timer_write) (uint32_t ch, uint32_t mode, uint32_t freq, uint32_t dc);    /*!< Set the configuration of pwm_timer*/
    int32_t (*pwm_timer_read) (uint32_t ch, uint32_t *mode, uint32_t *freq, uint32_t *dc);  /*!< Read the configuration of pwm_timer*/
} DEV_PWM_TIMER, * DEV_PWM_TIMER_PTR;
```

PWM - API (pwm_timer_open & pwm_timer_close)

```
typedef struct dev_pwm_timer {
    DEV_PWM_TIMER_INFO pwm_timer_info;              /*!< PWM_TIMER device information */
    int32_t (*pwm_timer_open) (void);               /*!< Open pwm_timer device */
    int32_t (*pwm_timer_close) (void);              /*!< Close pwm_timer device */
    int32_t (*pwm_timer_control) (uint32_t ch, uint32_t cmd, void *param2); /*!< Control pwm_timer device */
    int32_t (*pwm_timer_write) (uint32_t ch, uint32_t mode, uint32_t freq, uint32_t dc);    /*!< Set the configuration of pwm_timer*/
    int32_t (*pwm_timer_read) (uint32_t ch, uint32_t *mode, uint32_t *freq, uint32_t *dc);  /*!< Read the configuration of pwm_timer*/
} DEV_PWM_TIMER, * DEV_PWM_TIMER_PTR;
```

1. pwm_timer_open :
   Open pwm_timer device

   EX :

```
DEV_PWM_TIMER_PTR pwm_timer_0 = pwm_timer_get_dev(DW_PWM_TIMER_0_ID);
pwm_timer_0->pwm_timer_open();
```

2. Pwm_timer_close :
   Close pwm_timer device

   EX :

```
pwm_timer_0->pwm_timer_close();
```

PWM - API (pwm_timer_control)

```
typedef struct dev_pwm_timer {
    DEV_PWM_TIMER_INFO pwm_timer_info;              /*!< PWM_TIMER device information */
    int32_t (*pwm_timer_open) (void);               /*!< Open pwm_timer device */
    int32_t (*pwm_timer_close) (void);              /*!< Close pwm_timer device */
    int32_t (*pwm_timer_control) (uint32_t ch, uint32_t cmd, void *param2); /*!< Control pwm_timer device */
    int32_t (*pwm_timer_write) (uint32_t ch, uint32_t mode, uint32_t freq, uint32_t dc);    /*!< Set the configuration of pwm_timer*/
    int32_t (*pwm_timer_read) (uint32_t ch, uint32_t *mode, uint32_t *freq, uint32_t *dc);  /*!< Read the configuration of pwm_timer*/
} DEV_PWM_TIMER, * DEV_PWM_TIMER_PTR;
```

3. pwm_timer_control :
   Control an pwm_timer device [channel number: ch] by ctrl_cmd, with passed param.

```
#define PWM_TIMER0    IO3     0
#define PWM_TIMER1    IO5     1
#define PWM_TIMER2    IO6     2
#define PWM_TIMER3    IO9     3
#define PWM_TIMER4    IO10    4
#define PWM_TIMER5    IO11    5
```

```
#define PWM_TIMER_CMD_SET_CFG          DEV_SET_SYSCMD(0)

#define PWM_TIMER_CMD_GET_CFG          DEV_SET_SYSCMD(1)

#define PWM_TIMER_CMD_ENA_CH           DEV_SET_SYSCMD(2)

#define PWM_TIMER_CMD_DIS_CH           DEV_SET_SYSCMD(3)

#define PWM_TIMER_CMD_SET_ISR          DEV_SET_SYSCMD(4)

#define PWM_TIMER_CMD_GET_ISR          DEV_SET_SYSCMD(5)

#define PWM_TIMER_CMD_DIS_ISR          DEV_SET_SYSCMD(6)

#define PWM_TIMER_CMD_ENA_ISR          DEV_SET_SYSCMD(7)
```

PWM - API (pwm_timer_write)

```
typedef struct dev_pwm_timer {
    DEV_PWM_TIMER_INFO pwm_timer_info;              /*!< PWM_TIMER device information */
    int32_t (*pwm_timer_open) (void);               /*!< Open pwm_timer device */
    int32_t (*pwm_timer_close) (void);              /*!< Close pwm_timer device */
    int32_t (*pwm_timer_control) (uint32_t ch, uint32_t cmd, void *param2); /*!< Control pwm_timer device */
    int32_t (*pwm_timer_write) (uint32_t ch, uint32_t mode, uint32_t freq, uint32_t dc);    /*!< Set the configuration of pwm_timer*/
    int32_t (*pwm_timer_read) (uint32_t ch, uint32_t *mode, uint32_t *freq, uint32_t *dc);  /*!< Read the configuration of pwm_timer*/
} DEV_PWM_TIMER, * DEV_PWM_TIMER_PTR;
```

4. pwm_timer_write :
   Set the configuration of pwm_timer
   param :
   - uint32_t ch : channel to write
   - uint32_t mode : DEV_PWM_TIMER_MODE_PWM
   - uint32_t freq : 100000 (Hz)
   - uint32_t dc : duty cycle (0~100%)

```
#define PWM_TIMER0    IO3     0
#define PWM_TIMER1    IO5     1
#define PWM_TIMER2    IO6     2
#define PWM_TIMER3    IO9     3
#define PWM_TIMER4    IO10    4
#define PWM_TIMER5    IO11    5
```

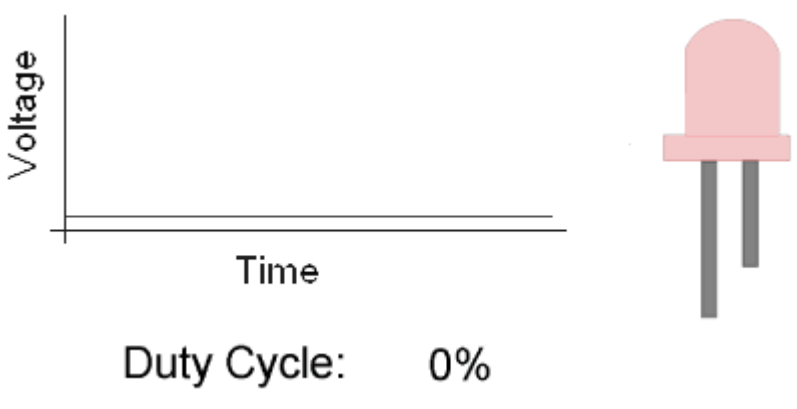
EX :

```
pwm_timer_0->pwm_timer_write(PWM_TIMER0, DEV_PWM_TIMER_MODE_PWM, 100000, 50);
```

```
typedef enum dev_pwm_timer_mode {
    DEV_PWM_TIMER_MODE_CLOSE = 0,   /*!< mode close */
    DEV_PWM_TIMER_MODE_TIMER = 1,   /*!< mode timer */
    DEV_PWM_TIMER_MODE_PWM = 2      /*!< mode pwm */
} DEV_PWM_TIMER_MODE, *DEV_PWM_TIMER_MODE_PTR;
```

PWM - API (pwm_timer_write example)

Lighten and Darken




```c
uint32_t dc;
while(1)
{
    for(dc=0;dc<100;dc++)
    {
        pwm_timer_0->pwm_timer_write(PWM_TIMER0, DEV_PWM_TIMER_MODE_PWM, 100000, dc);
        board_delay_ms(7, 0);
    }
    for(dc=100;dc>0;dc--)
    {
        pwm_timer_0->pwm_timer_write(PWM_TIMER0, DEV_PWM_TIMER_MODE_PWM, 100000, dc);
        board_delay_ms(7, 0);
    }
}
```

PWM - API (pwm_timer_read)

```
typedef struct dev_pwm_timer {
    DEV_PWM_TIMER_INFO pwm_timer_info;              /*!< PWM_TIMER device information */
    int32_t (*pwm_timer_open) (void);               /*!< Open pwm_timer device */
    int32_t (*pwm_timer_close) (void);              /*!< Close pwm_timer device */
    int32_t (*pwm_timer_control) (uint32_t ch, uint32_t cmd, void *param2); /*!< Control pwm_timer device */
    int32_t (*pwm_timer_write) (uint32_t ch, uint32_t mode, uint32_t freq, uint32_t dc);    /*!< Set the configuration of pwm_timer*/
    int32_t (*pwm_timer_read) (uint32_t ch, uint32_t *mode, uint32_t *freq, uint32_t *dc);  /*!< Read the configuration of pwm_timer*/
} DEV_PWM_TIMER, * DEV_PWM_TIMER_PTR;
```

Error ?

5. pwm_timer_read :
   Read the configuration of pwm_timer
   param :
   - uint32_t ch : channel to read
   - uint32_t mode : save mode to *mode
   - uint32_t freq : save freq to *freq
   - uint32_t dc : save duty cycle to *dc

```
#define PWM_TIMER0    IO3     0
#define PWM_TIMER1    IO5     1
#define PWM_TIMER2    IO6     2
#define PWM_TIMER3    IO9     3
#define PWM_TIMER4    IO10    4
#define PWM_TIMER5    IO11    5
```

PWM - Summary

```
typedef struct dev_pwm_timer {
    DEV_PWM_TIMER_INFO pwm_timer_info;              /*!< PWM_TIMER device information */
    int32_t (*pwm_timer_open) (void);               /*!< Open pwm_timer device */
    int32_t (*pwm_timer_close) (void);              /*!< Close pwm_timer device */
    int32_t (*pwm_timer_control) (uint32_t ch, uint32_t cmd, void *param2); /*!< Control pwm_timer device */
    int32_t (*pwm_timer_write) (uint32_t ch, uint32_t mode, uint32_t freq, uint32_t dc);    /*!< Set the configuration of pwm_timer*/
    int32_t (*pwm_timer_read) (uint32_t ch, uint32_t *mode, uint32_t *freq, uint32_t *dc);  /*!< Read the configuration of pwm_timer*/
} DEV_PWM_TIMER, * DEV_PWM_TIMER_PTR;
```

```
DEV_PWM_TIMER_PTR pwm_timer_0 = pwm_timer_get_dev(DW_PWM_TIMER_0_ID);
pwm_timer_0->pwm_timer_open();
```

```
pwm_timer_0->pwm_timer_close();
```

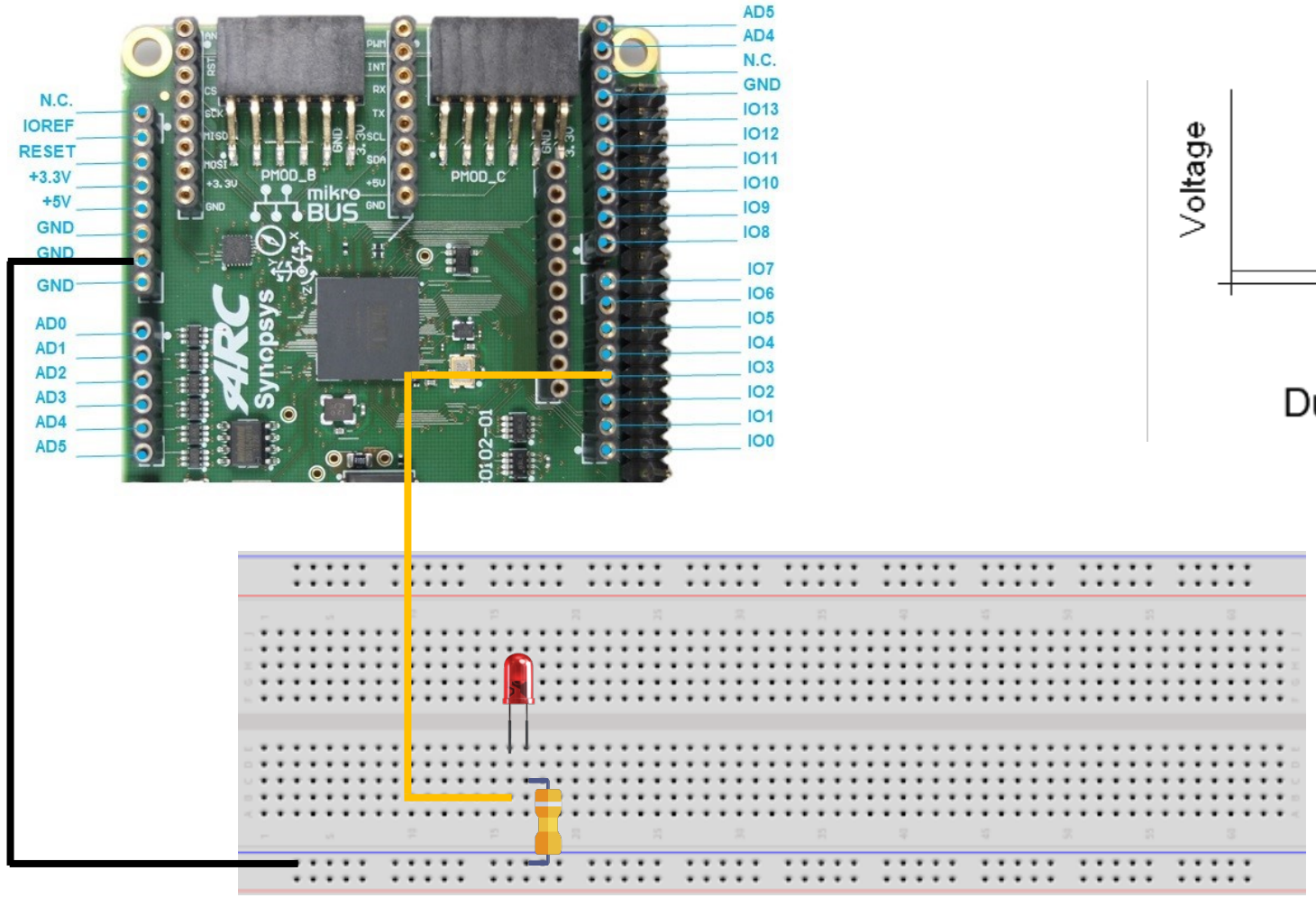
```
pwm_timer_0->pwm_timer_write(PWM_TIMER0, DEV_PWM_TIMER_MODE_PWM, 100000, 50);
```

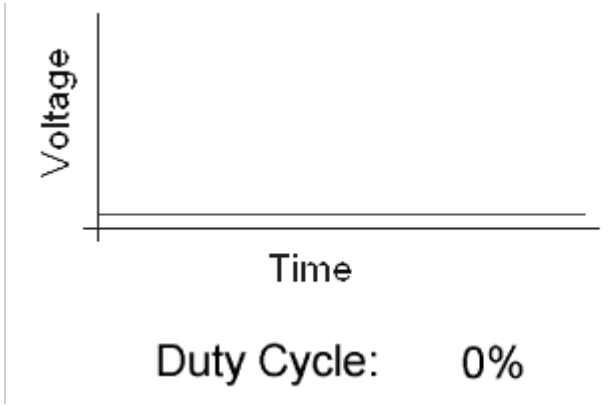
#define PWM_TIMER0	IO3	0
#define PWM_TIMER1	IO5	1
#define PWM_TIMER2	IO6	2
#define PWM_TIMER3	IO9	3
#define PWM_TIMER4	IO10	4
#define PWM_TIMER5	IO11	5

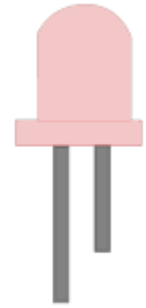
freq = 100000 (Hz)
dc = 0~100 (%)

```
#include "embARC.h"
#include "embARC_debug.h"

void arduino_pin_init(void)
{
    io_arduino_config(ARDUINO_PIN_3, ARDUINO_PWM, IO_PINMUX_ENABLE);//pwm timer ch0
    io_arduino_config(ARDUINO_PIN_5, ARDUINO_PWM, IO_PINMUX_ENABLE);//pwm timer ch1
    io_arduino_config(ARDUINO_PIN_6, ARDUINO_PWM, IO_PINMUX_ENABLE);//pwm timer ch2
    io_arduino_config(ARDUINO_PIN_9, ARDUINO_PWM, IO_PINMUX_ENABLE);//pwm timer ch3
    io_arduino_config(ARDUINO_PIN_10, ARDUINO_PWM, IO_PINMUX_ENABLE);//pwm timer ch4
    io_arduino_config(ARDUINO_PIN_11, ARDUINO_PWM, IO_PINMUX_ENABLE);//pwm timer ch5
}

int main(void)
{
    /* change pins settings from gpio to pwm */
    arduino_pin_init();

    // ...

    return E_SYS
}
```

PWM - Example1 (Breathing light)







PWM - Example2 (Colorful light)

Color Link :
https://www.toolskk.com/color

EX :

#f7bbd9 → (15*16+7, 11*16+11, 13*16+9) → (247, 187, 217)

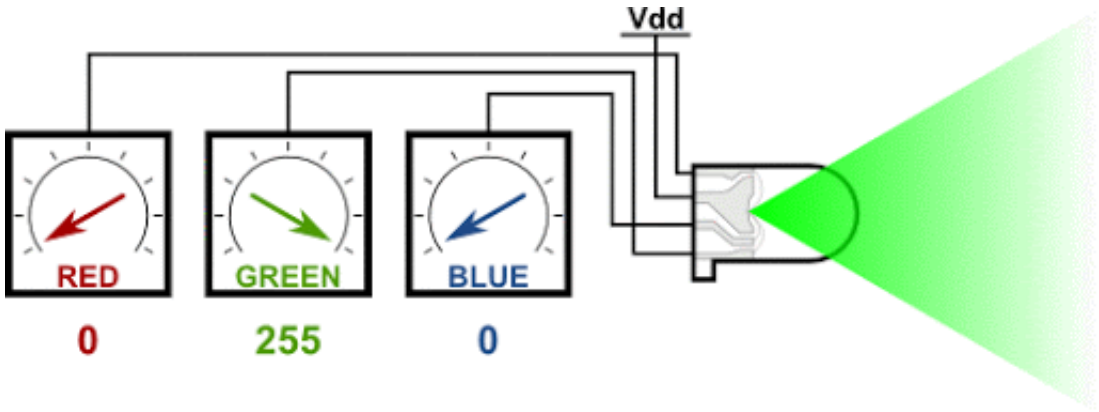
R G B

DC :

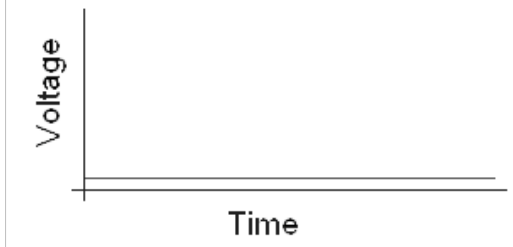
R → $247 \times \frac{100}{255} \cong 96.8$ → 97%

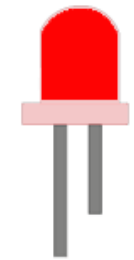
G → $187 \times \frac{100}{255} \cong 73.3$ → 73%

B → $217 \times \frac{100}{255} \cong 85.0$ → 85%

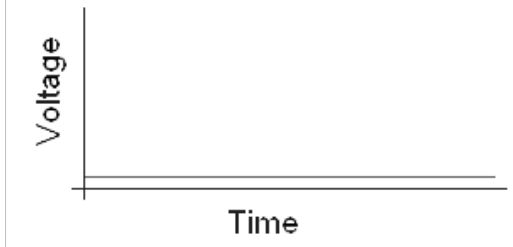


Red Duty Cycle: 100%



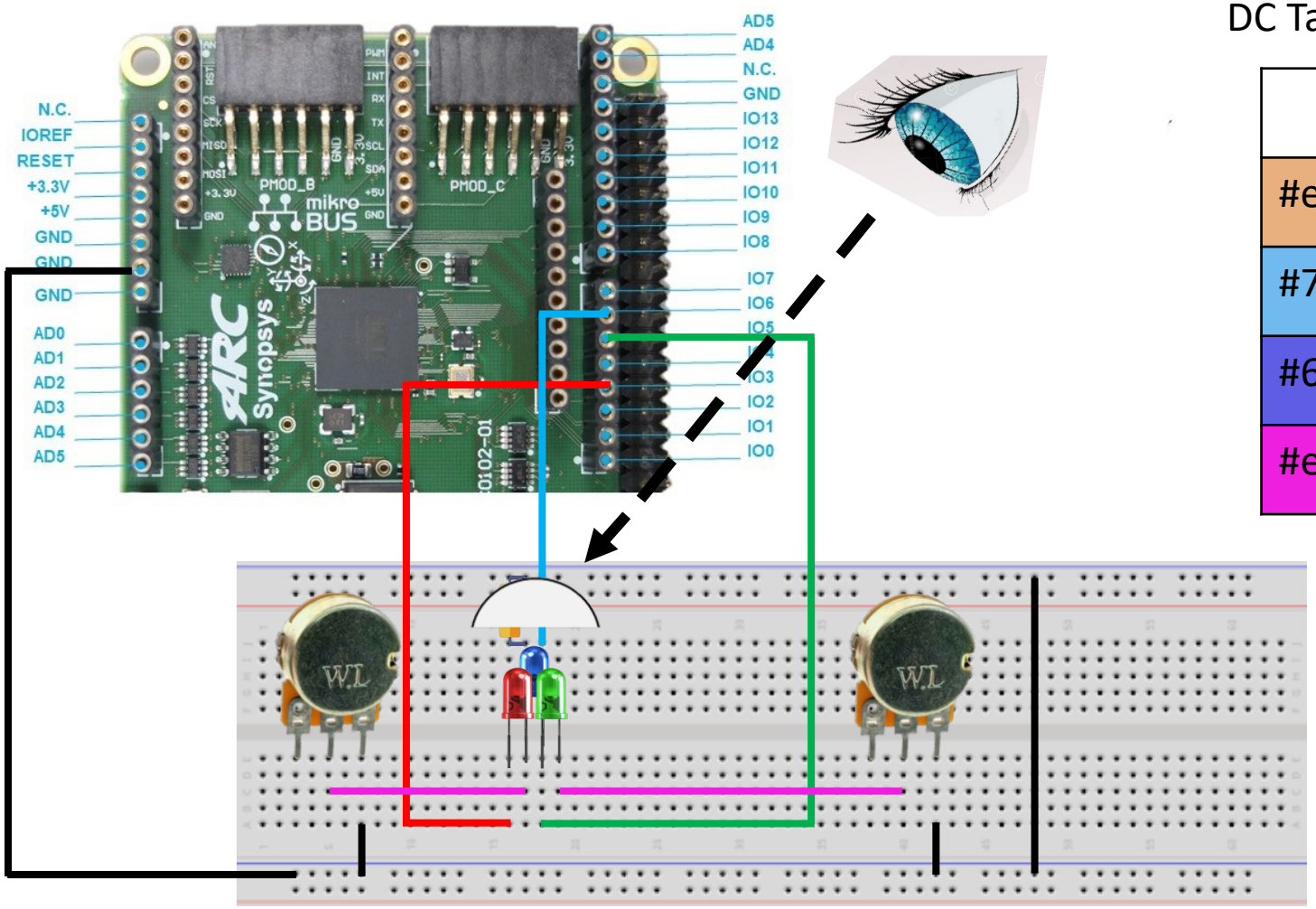


Green Duty Cycle: 0%



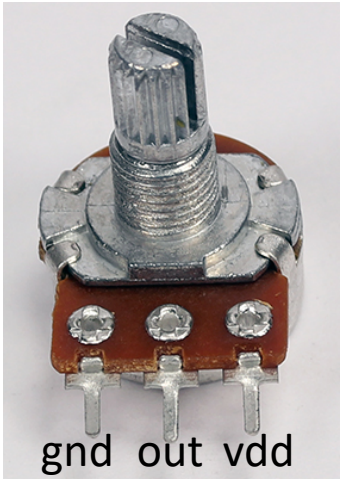
Blue Duty Cycle: 0%

PWM - Example2 (Colorful light)



DC Table :

	R	G	B
#e7b07e	91	69	49
#71baef	44	73	94
#605de5	38	36	90
#ee20df	93	13	87



PWM - LAB

Components :

(1) Red(R), Green(G), Blue(B) LEDs
(2) 2 * Variable resistance (for R & G)
(3) 1 * Special Mask

Function :

-> -> -> -> -> ->

each for 2 times

DC Table :

	R	G	B
Red	100	0	0
Orange	100	40	0
Yellow	100	100	0
Green	0	100	0
Blue	0	0	100
Purple	100	0	100
White	100	100	100