Lab 1: Familiar with MKR 1010

- Lab return must be in a pdf format and follow the structure shown in the first lecture

Guidance:

1. Connect Arduino via USB

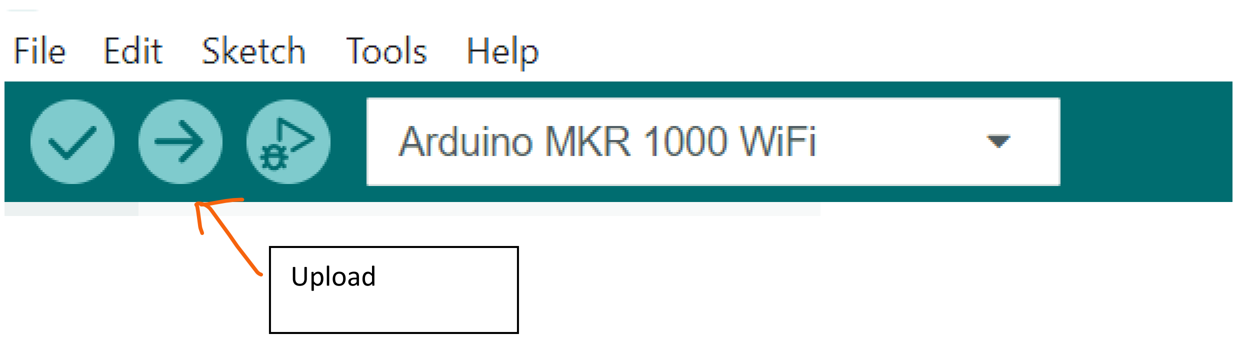
- Use a **USB cable** (Micro-USB for MKR 1010

2. **Configure the IDE**

- Open Arduino IDE
- Go to Tools > Board > Select your Arduino - MKR WiFi 1010
- Go to Tools > Port > Select the correct COM port

3. Write a code

4. Upload the Code

- Click the Upload button (right arrow icon in top-left)
- 

- Wait for message: Done uploading.

For opening terminal: Tools -> Serial Monitor

Microcontroller SAM21 datasheet: https://ww1.microchip.com/downloads/en/DeviceDoc/SAM-D21DA1-Family-Data-Sheet-DS40001882G.pdf

Exercise 1.1: Familiar with Digital & Analog I/O Basics (0.5p)

1. Wire the LED on D6 and the potentiometer on A1
2. Write a sketch that:
   - Blinks the LED: 500 ms ON / 500 ms OFF (digitalWrite on D6).

    - Reads analogRead(A1) once per second and prints the raw value to Serial at 9600 baud.
3. While the LED is blinking, rotate the pot and observe the readings (0–4095 on SAMD21-based MKR1010). Print out the read values into a terminal

Exercise 1.2: Familiar with Wi-Fi library (0.5p)

1. Install WiFiNINA library for MKR1010. Location of the installed library can be in Documents\Arduino\libraries
2. Use the code below. However, there are a few errors in the code below – Fix them and upload it to Arduino MKR 1010 successfully. Note: you need to go to the location of the installed library to see how functions and classes related to Wi-Fi are implemented.
3. Provide comments for every line of your fixed code

Note: The code will scan available networks every 30 seconds and print out all available networks. Then, it prints out the total number of available networks

```
#include <WiFiNINA.h>

void setup() {

  Serial.begin(9600);

}

void loop() {

  if (WiFi.status() == WL_NO_MODULE) {

    Serial.println("WiFi module not found!");

}

  Serial.println("Scanning available networks...");

  uint8_t numNetworks = WiFi.scanNetworks();

  for (uint8_t = 0; i < numNetworks; i++) {

    Serial.print("Network: ");
    Serial.print(WiFi.SSID(i));
    Serial.print(" | Signal Strength: ");
    Serial.print(WiFi.RSSI(i));
```

```
    Serial.print(" dBm | Encryption: ");
    Serial.println(WiFi.encryptionType(i));

  }

delay(10000);
delay(10000);
delay(10000);

}
```

Exercise 1.3: Making a basic calculator with MKR1010 (0.5p)

Note: You can consider using this code line for your code:

```
String input = Serial.readStringUntil('\n');
```

Explanation:

- The Arduino is connected to your computer through a USB cable.
- When you type something in the Serial Monitor and press Enter, those characters travel to the MKR1010.
- This line reads everything you typed until it finds a "newline" character ('\n'), which is added when you hit Enter.

You also consider this code line for your code

```
sscanf(input.c_str(), "%f %c %f", &num1, &op, &num2);
```

Explain:

1. input.c_str()

input is an Arduino String.

.c_str() turns it into a C-style string (a sequence of characters ending with \0), which sscanf needs.

2. "%f %c %f"

   This is the format pattern telling sscanf what to look for in the string:

   - %f → read a floating-point number (stored in num1)
   - %c → read a single character (stored in op)
   - %f → read another floating-point number (stored in num2)

This code line will read input string and store extracted values into variables

Proceture Step:

A . The MKR will ask an user to provide the numbers

B. An user will provide like 8 * 5 or 8 – 3 or 8/4 or 1 + 4

C. MKR will act as a simple calculator to provide the result from the provided numbers and operator

D. Print out the results to the terminal and continue the next loop to ask for a new operation

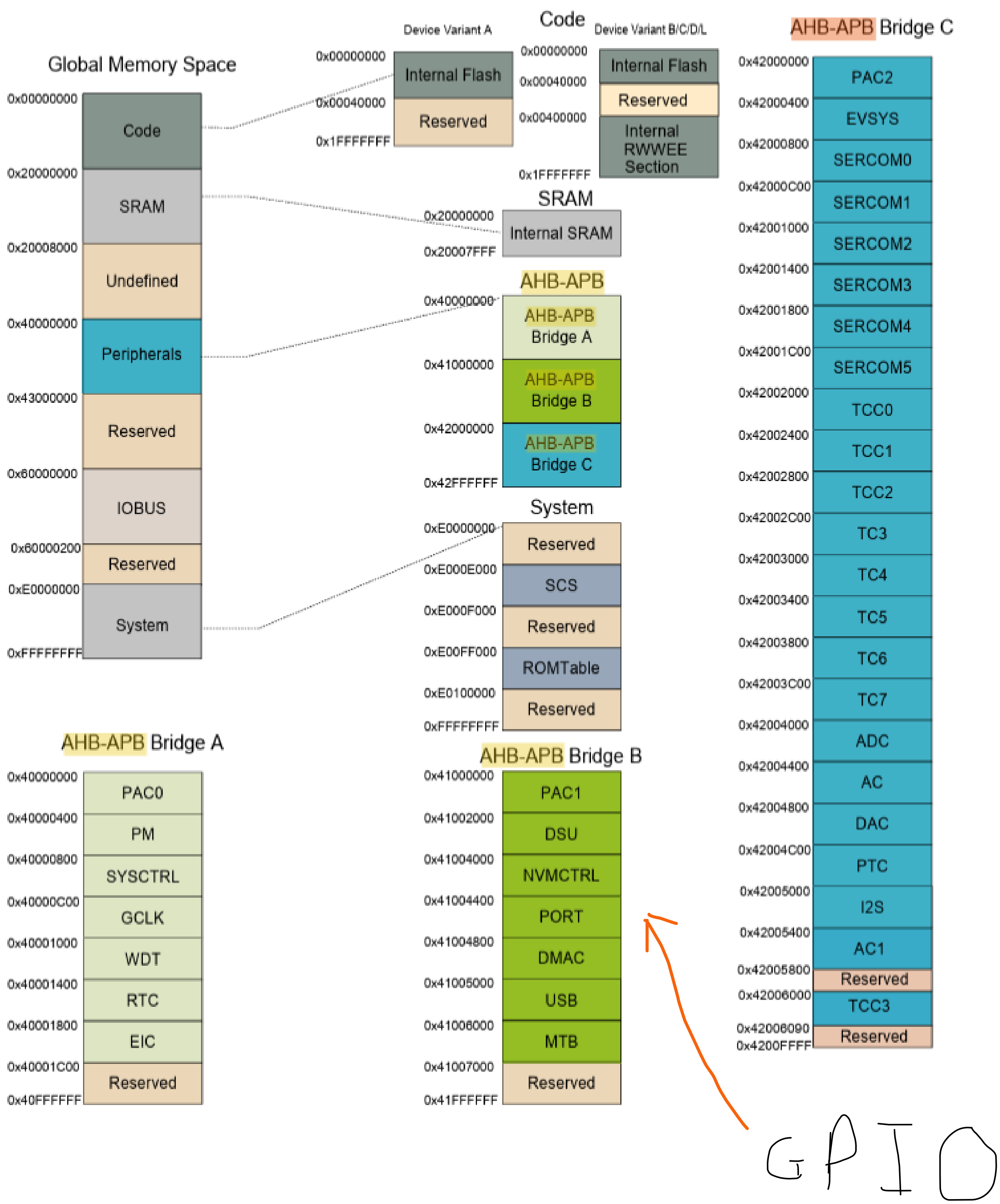
Exercise 1.4:

Below is bare metal bitwise programing

```
#include "sam.h"
#define LED_PIN   20   // D6 → PA20 on MKR1010
#define INPUT_PIN 10   // D2 → PA10

void setup() {
  // Enable the APB clock for the PORT module
  PM->APBBMASK.reg |= PM_APBBMASK_PORT;

  // Configure LED_PIN as OUTPUT (set bit in DIRSET)
  PORT->Group[0].DIRSET.reg = (1 << LED_PIN);

  // Configure INPUT_PIN as INPUT (clear bit in DIRCLR)
  PORT->Group[0].DIRCLR.reg = (1 << INPUT_PIN);
}

void loop() {
  // Turn LED ON
```

```
  PORT->Group[0].OUTSET.reg = (1 << LED_PIN);
  delay(500);

  // Turn LED OFF
  PORT->Group[0].OUTCLR.reg = (1 << LED_PIN);
  delay(500);
}
```

Note: The code above turns on and off built-in LED every 500ms

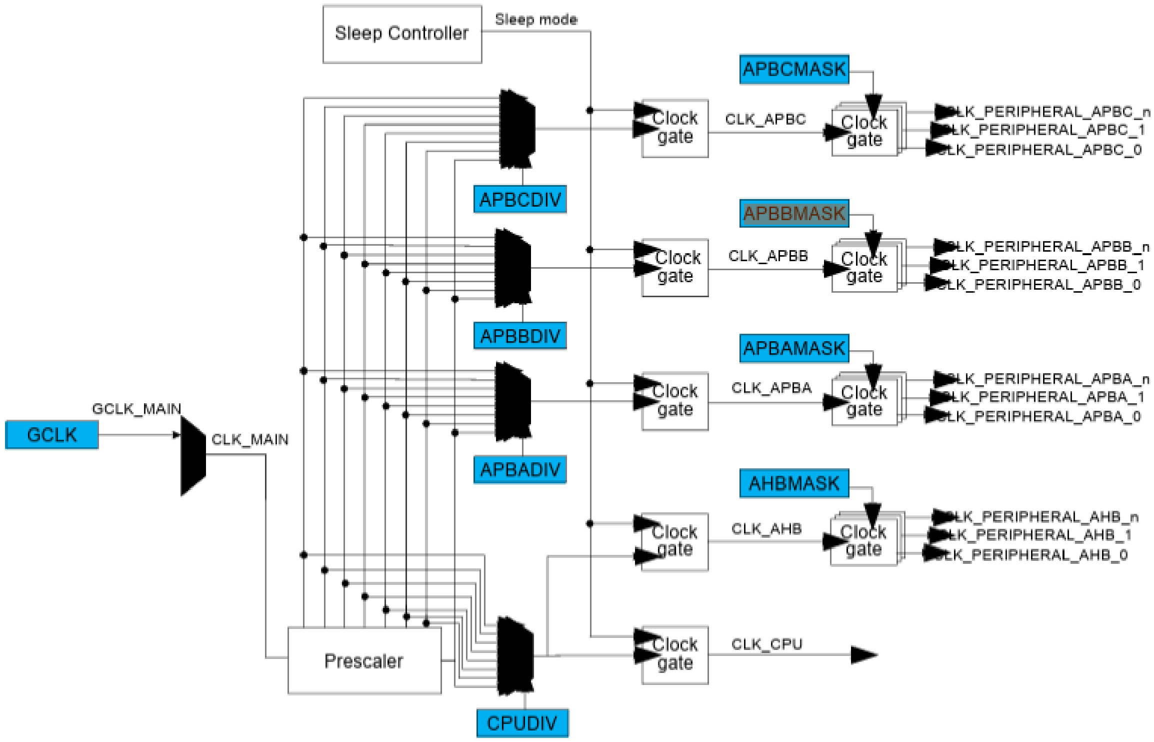


Global Memory Space
0x00000000
Code
0x20000000
SRAM
0x20008000
Undefined
0x40000000
Peripherals
0x43000000
Reserved
0x60000000
IOBUS
0x60000200
Reserved
0xE0000000
System
0xFFFFFFFF
Code
Device Variant A
0x00000000
Internal Flash
0x00040000
Reserved
0x1FFFFFFF
Device Variant B/C/D/L
0x00000000
Internal Flash
0x00040000
Reserved
0x00400000
Internal RWWEE Section
0x1FFFFFFF
SRAM
0x20000000
Internal SRAM
0x20007FFF
AHB-APB
0x40000000
AHB-APB Bridge A
0x41000000
AHB-APB Bridge B
0x42000000
AHB-APB Bridge C
0x42FFFFFF
System
0xE0000000
Reserved
0xE000E000
SCS
0xE000F000
Reserved
0xE00FF000
ROMTable
0xE0100000
Reserved
0xFFFFFFFF
AHB-APB Bridge A
0x40000000
PAC0
0x40000400
PM
0x40000800
SYSCTRL
0x40000C00
GCLK
0x40001000
WDT
0x40001400
RTC
0x40001800
EIC
0x40001C00
Reserved
0x40FFFFFF
AHB-APB Bridge B
0x41000000
PAC1
0x41002000
DSU
0x41004000
NVMCTRL
0x41004400
PORT
0x41004800
DMAC
0x41005000
USB
0x41006000
MTB
0x41007000
Reserved
0x41FFFFFF
AHB-APB Bridge C
0x42000000
PAC2
0x42000400
EVSYS
0x42000800
SERCOM0
0x42000C00
SERCOM1
0x42001000
SERCOM2
0x42001400
SERCOM3
0x42001800
SERCOM4
0x42001C00
SERCOM5
0x42002000
TCC0
0x42002400
TCC1
0x42002800
TCC2
0x42002C00
TC3
0x42003000
TC4
0x42003400
TC5
0x42003800
TC6
0x42003C00
TC7
0x42004000
ADC
0x42004400
AC
0x42004800
DAC
0x42004C00
PTC
0x42005000
I2S
0x42005400
AC1
0x42005800
Reserved
0x42006000
TCC3
0x42006090
Reserved
0x4200FFFF
GPIO

16.7 Register Summary

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00	CTRL	7:0								
0x01	SLEEP	7:0							IDLE[1:0]	
0x02 ... 0x07	Reserved									
0x08	CPUSEL	7:0						CPUDIV[2:0]		
0x09	APBASEL	7:0						APBADIV[2:0]		
0x0A	APBBSEL	7:0						APBBDIV[2:0]		
0x0B	APBCSEL	7:0						APBCDIV[2:0]		
0x0C ... 0x13	Reserved									
0x14	AHBMASK	7:0		USB	DMAC	NVMCTRL	DSU	HPB2	HPB1	HPB0
		15:8								
		23:16								
		31:24								
0x18	APBAMASK	7:0		EIC	RTC	WDT	GCLK	SYSCTRL	PM	PAC0
		15:8								
		23:16								
		31:24								
0x1C	APBBMASK	7:0			USB	DMAC	PORT	NVMCTRL	DSU	PAC1
		15:8								
		23:16								
		31:24								
0x20	APBCMASK	7:0	SERCOM5	SERCOM4	SERCOM3	SERCOM2	SERCOM1	SERCOM0	EVSYS	PAC2
		15:8	TC7	TC6	TC5	TC4	TC3	TCC2	TCC1	TCC0
		23:16			AC1	I2S	PTC	DAC	AC	ADC
		31:24								TCC3
0x24 ... 0x33	Reserved									
0x34	INTENCLR	7:0								CKRDY
0x35	INTENSET	7:0								CKRDY
0x36	INTFLAG	7:0								CKRDY
0x37	Reserved									
0x38	RCAUSE	7:0		SYST	WDT	EXT		BOD33	BOD12	POR

Figure 16-2. Synchronous Clock Selection and Prescaler



16.8.9 APBB Mask

Name: APBBMASK
Offset: 0x1C
Reset: 0x0000007F
Property: Write-Protected

Bit	31	30	29	28	27	26	25	24
Access								
Reset								

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
Access								
Reset								

Bit	7	6	5	4	3	2	1	0
			USB	DMAC	PORT	NVMCTRL	DSU	PAC1
Access			R/W	R/W	R/W	R/W	R/W	R/W
Reset			1	1	1	1	1	1

Bit 5 – USB USB APB Clock Enable

Value	Description
0	The APBB clock for the USB is stopped.
1	The APBB clock for the USB is enabled.

Bit 4 – DMAC DMAC APB Clock Enable

Value	Description
0	The APBB clock for the DMAC is stopped.
1	The APBB clock for the DMAC is enabled.

Bit 3 – PORT PORT APB Clock Enable

Value	Description
0	The APBB clock for the PORT is stopped.
1	The APBB clock for the PORT is enabled.

Bit 2 – NVMCTRL NVMCTRL APB Clock Enable

Value	Description
0	The APBB clock for the NVMCTRL is stopped.
1	The APBB clock for the NVMCTRL is enabled.

Bit 1 – DSU DSU APB Clock Enable

Value	Description
0	The APBB clock for the DSU is stopped.
1	The APBB clock for the DSU is enabled.

Bit 0 – PAC1 PAC1 APB Clock Enable

Value	Description
0	The APBB clock for the PAC1 is stopped.
1	The APBB clock for the PAC1 is enabled.

Let's explore the code above

A)

For a SAM chip, you need to enable the peripheral clock for using PORTs

PM->APBBMASK.reg |= PM_APBBMASK_PORT;

This is equivalent with PM->APBBMASK.reg |= (1<<3)

PM

- Refers to the **Power Manager** peripheral.
- The Power Manager controls **which modules (peripherals)** receive a clock signal.

Think of it as a master power switch for every subsystem (ADC, timers, GPIO, etc.).

Element	Description
`PM`	Power Manager peripheral
`APBBMASK`	Register controlling which modules on APBB bus get a clock
`PORT`	GPIO peripheral (controls all pins)
`PM_APBBMASK_PORT`	Bit mask for enabling PORT module clock
Effect	Enables GPIO functionality by turning on its clock source

```
/* -------- PM_APBBMASK : (PM Offset: 0x1C) (R/W 32) APBB Mask -------- */
#if !(defined(__ASSEMBLY__) || defined(__IAR_SYSTEMS_ASM__))
typedef union {
  struct {
    uint32_t PAC1_:1;          /*!< bit:      0  PAC1 APB Clock Enable            */
    uint32_t DSU_:1;           /*!< bit:      1  DSU APB Clock Enable             */
    uint32_t NVMCTRL_:1;       /*!< bit:      2  NVMCTRL APB Clock Enable         */
    uint32_t PORT_:1;          /*!< bit:      3  PORT APB Clock Enable            */
    uint32_t DMAC_:1;          /*!< bit:      4  DMAC APB Clock Enable            */
    uint32_t USB_:1;           /*!< bit:      5  USB APB Clock Enable             */
    uint32_t HMATRIX_:1;       /*!< bit:      6  HMATRIX APB Clock Enable         */
    uint32_t :25;              /*!< bit:  7..31  Reserved                         */
  } bit;                       /*!< Structure used for bit  access                */
  uint32_t reg;                /*!< Type      used for register access            */
} PM_APBBMASK_Type;
#endif /* !(defined(__ASSEMBLY__) || defined(__IAR_SYSTEMS_ASM__)) */
```

```c
#define PM_APBBMASK_OFFSET          0x1C        /**< \brief (PM_APBBMASK offset) APBB Mask */
#define PM_APBBMASK_RESETVALUE      0x0000007Ful /**< \brief (PM_APBBMASK reset_value) APBB Mask */

#define PM_APBBMASK_PAC1_Pos        0           /**< \brief (PM_APBBMASK) PAC1 APB Clock Enable */
#define PM_APBBMASK_PAC1            (0x1ul << PM_APBBMASK_PAC1_Pos)
#define PM_APBBMASK_DSU_Pos         1           /**< \brief (PM_APBBMASK) DSU APB Clock Enable */
#define PM_APBBMASK_DSU             (0x1ul << PM_APBBMASK_DSU_Pos)
#define PM_APBBMASK_NVMCTRL_Pos     2           /**< \brief (PM_APBBMASK) NVMCTRL APB Clock Enable */
#define PM_APBBMASK_NVMCTRL         (0x1ul << PM_APBBMASK_NVMCTRL_Pos)
#define PM_APBBMASK_PORT_Pos        3           /**< \brief (PM_APBBMASK) PORT APB Clock Enable */
#define PM_APBBMASK_PORT            (0x1ul << PM_APBBMASK_PORT_Pos)
#define PM_APBBMASK_DMAC_Pos        4           /**< \brief (PM_APBBMASK) DMAC APB Clock Enable */
#define PM_APBBMASK_DMAC            (0x1ul << PM_APBBMASK_DMAC_Pos)
#define PM_APBBMASK_USB_Pos         5           /**< \brief (PM_APBBMASK) USB APB Clock Enable */
#define PM_APBBMASK_USB             (0x1ul << PM_APBBMASK_USB_Pos)
#define PM_APBBMASK_HMATRIX_Pos     6           /**< \brief (PM_APBBMASK) HMATRIX APB Clock Enable */
#define PM_APBBMASK_HMATRIX         (0x1ul << PM_APBBMASK_HMATRIX_Pos)
#define PM_APBBMASK_MASK            0x0000007Ful /**< \brief (PM_APBBMASK) MASK Register */
```

This line **enables the peripheral clock** for the **PORT module** — the hardware block that controls the GPIO pins — on the **SAMD21** microcontroller (used in the MKR WiFi 1010).

Without this line, the **PORT registers (DIR, OUT, IN, etc.)** won't work because their **clock is disabled by default** to save power.

B)

```c
// Configure LED_PIN as OUTPUT (set bit in DIRSET)
PORT->Group[0].DIRSET.reg = (1 << LED_PIN);
```

`PORT`

- `PORT` is a **pointer to the PORT peripheral's register structure**.
- It's defined in `sam.h` (the SAMD21 device header).
- The SAMD21 has **one PORT controller** managing up to 2 groups:
  - `Group[0]` → Port A (PA00–PA31)
  - `Group[1]` → Port B (PB00–PB31)

So, `PORT->Group[0]` means we're accessing **Port A** registers.

`.DIRSET`

- Each group (Port A or B) has a register set for direction control:

- `DIR` → Data Direction Register (1 = output, 0 = input)
- `DIRSET` → Direction **Set** Register
- `DIRCLR` → Direction **Clear** Register
- `DIRTGL` → Direction **Toggle** Register

`DIRSET` is a **write-only** register used to set specific bits **to 1** without changing others.

`.reg`

- `.reg` accesses the **actual 32-bit register value**.
  This is how CMSIS represents memory-mapped registers in structs.

Register	**Access**	**Function**	**Effect**
`DIR`	R/W	Direction register	1 = output, 0 = input
`DIRSET`	W	Set direction bits to 1	Make pin output
`DIRCLR`	W	Clear direction bits to 0	Make pin input
`DIRTGL`	W	Toggle direction bits	Flip input/output

Task:

Connect a new LED to pin PB10-D4
Toggling a LED every second