LAB1 - GPIO Control Design

【PURPOSE】

- Understand the features and specifications of the NuMaker-PFM-M487 series processor.
- Understand the advantages of ARM® Cortex®-Mx series processors.
- Utilize the Keil uVision development tool software.

【BACKGROUND】

C Language

【Principle and illustrate】

I. Understand the NuMaker-PFM-M487 series processor.

- **Basic features and specifications:**

• Core
- ARM® Cortex™-M4 with DSP and FPU
- Max frequency of 72 MHz
- Operating voltage: 2.5V to 5.5V
- Temperature range: -40℃ ~ 105℃

• Memory
- 128/256 KB of Flash Memory
- 32 KB of SRAM
- Data Flash configurable

• Analog Peripheral
- One 12-bit, up to 16-ch 5MSPS SAR ADC
- Two 12-bit, 1 MSPS DAC
- Two rail-to-rail comparators
- Three operational amplifiers

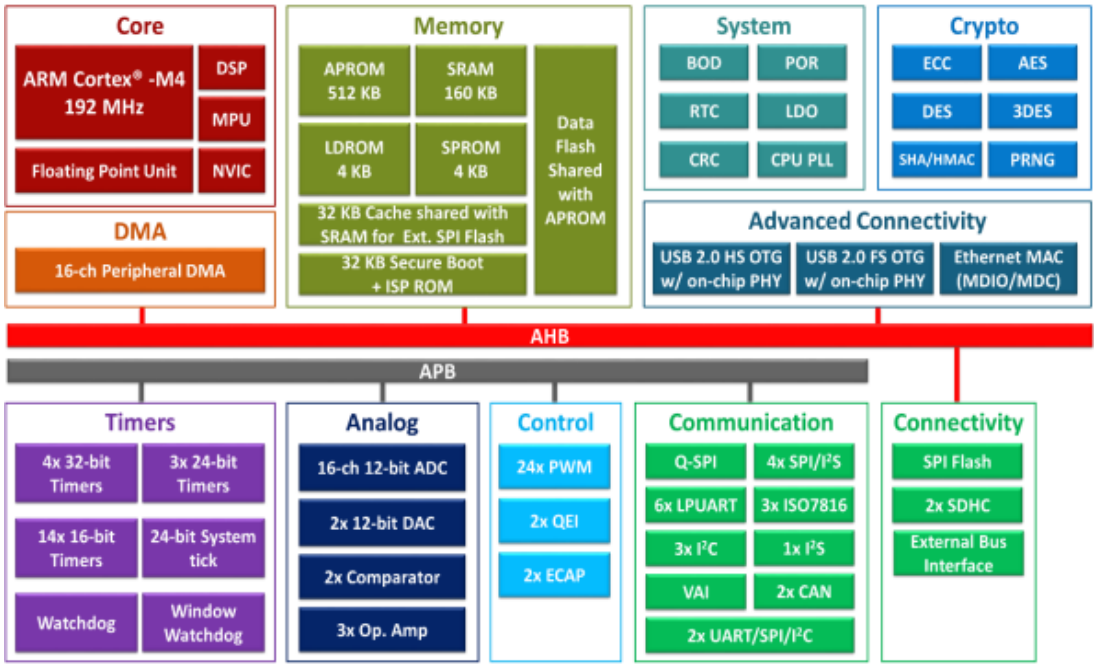
• Timer & PWM
- Four 32-bit timers
- 12 Enhanced PWM with twelve 16-bit timers
- 12 Basic PWM with two 16-bit timers
- One 24-bit count-down SysTick timer
- One watchdog timer
- One window watchdog timer

• Communication Interface
- Up to 6 low power UART interfaces
- Up to 3 ISO-7816 interfaces
- Three I²C interfaces
- One SPI Flash interface
- One Quad- SPI interface
- Up to 4 SPI / I²S interfaces
- One I²S interface
- Two USCI interfaces
- Two CAN 2.0B interfaces
- Two Secure Digital Host

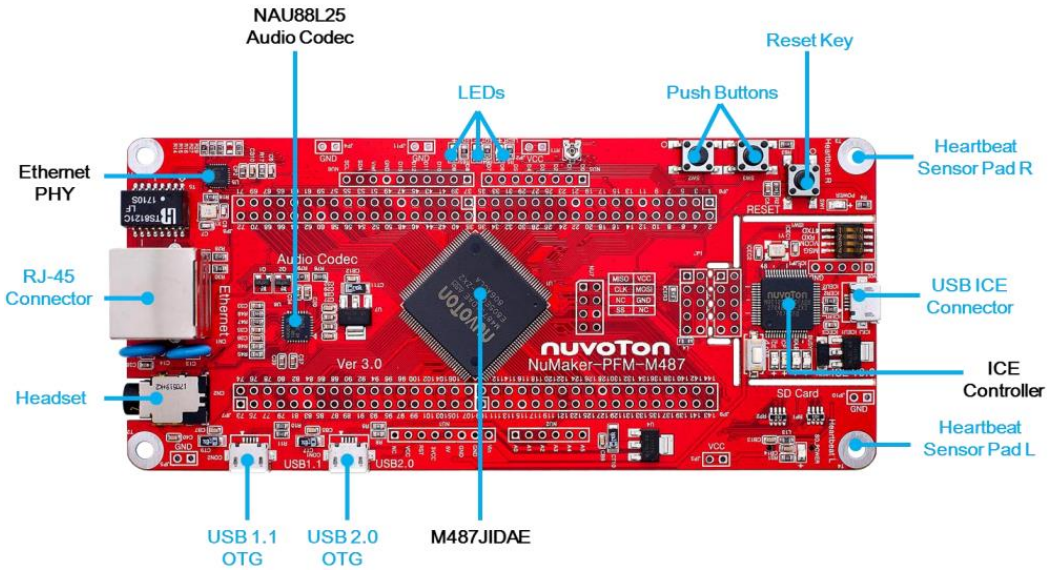
• Cryptography Accelerator
- ECC (P-521, B-571, K-571)
- AES-128, 192, 256
- DES/ 3DES
- SHA-160, 224, 256, 384, 512
- HMAC
- Random number generator

• Clocks
- 4 to 24 MHz crystal oscillator
- 32 kHz oscillator for RTC
- Internal 12 M/ 10K Hz RC oscillator
- Internal PLL up to 480 MHz

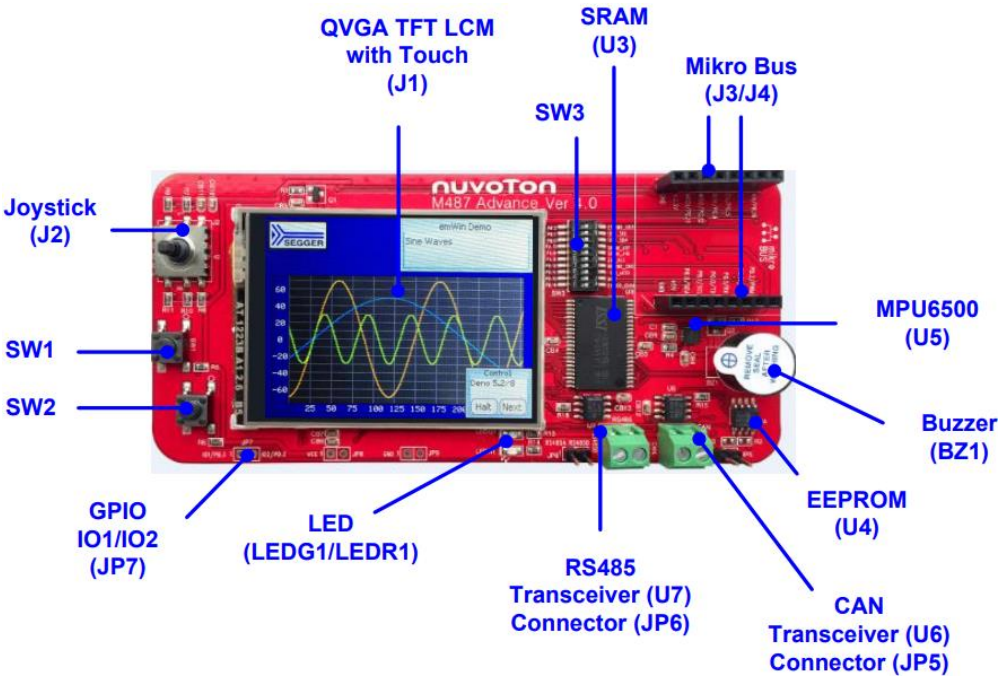
· **M487JIDAE functional block diagram：**



· **M487JIDAE Peripheral hardware：**



- **NuMaker M487 Advance Peripheral hardware:**

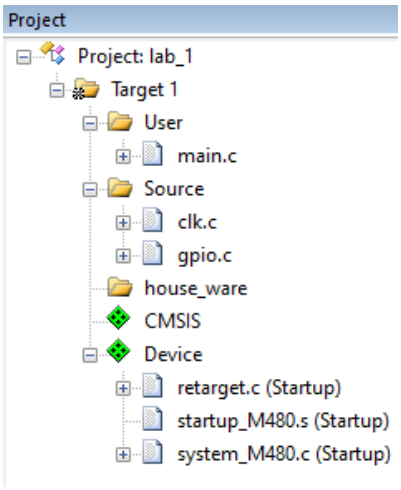


II. Keil uVision Usage

See power point for details

III. Project Group environment

In our project directory, it will be composed of three Groups.。



1. source

   The “source” folder is primarily used to store source files(.c) from Nuvoton Technology Corp., where each source file defines every function in the corresponding header file (.h). If we want to understand how a specific function from the official source is executed, we can refer to the source file.

2. House_ware

   The “House_ware” folder contains header and source files that we have written ourselves. We define the functions that we frequently use. To distinguish them from official files, we place our own designed programs (including .c and .h files) in this group.

3. User

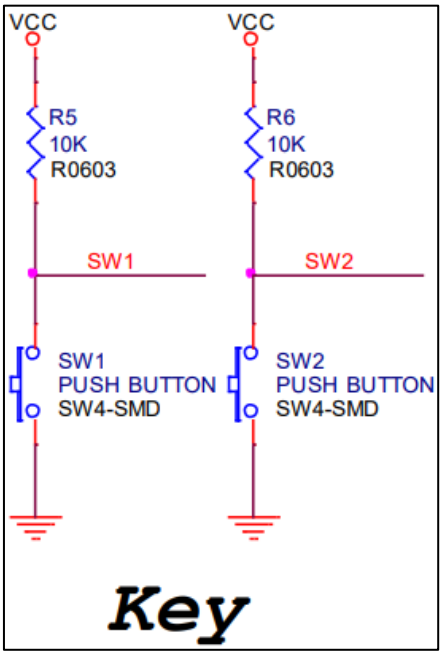
   The “User” folder will only contain the main.c file, which is the main program we execute. We only need to include the header files from the first two groups in the main program. During experiments, we only need to modify main.c to achieve the required functionality.

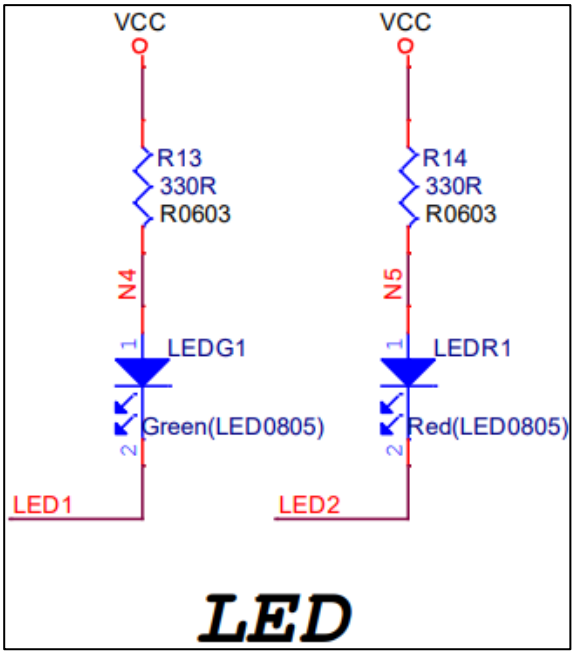
IV. GPIO Control Design

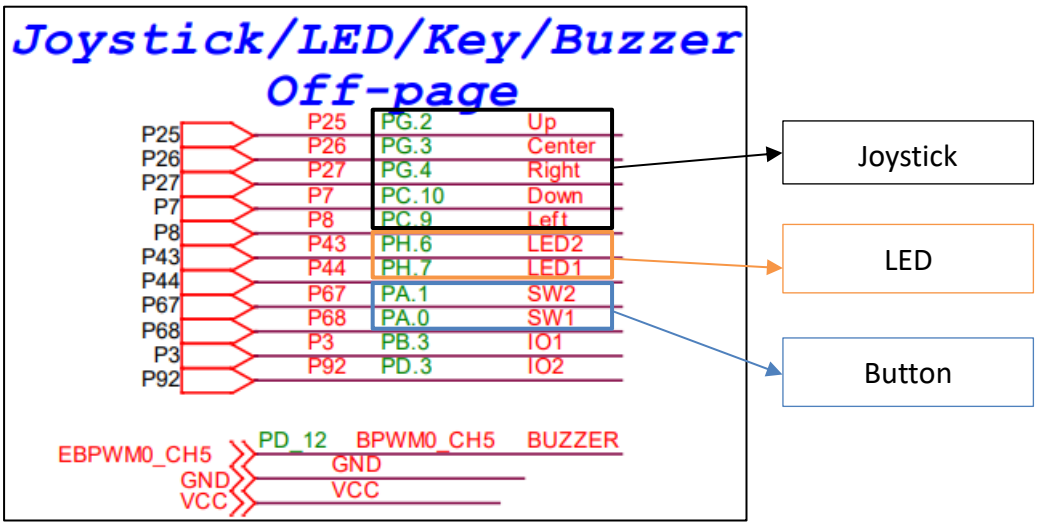
Example:

The experiment code shows the LED control using buttons (keys). As shown in the previous page of the NuMaker M487 Advance, SW1 controls LEDG1, and SW2 controls LEDR1. When the button is pressed, the corresponding LED lights up.

To control the LED with a button, The first thing is understand the GPIO pins corresponding to the button and LED. This information can be found by referring to the manual provided by Nuvoton, as illustrated in the diagram below.







From the image, it is observed that PA.0 (button) and PA.1 must be set as inputs, where the button is pressed when the value is 0 and released when it is 1. PH.6 (LED) and PH.7 must be set as outputs, where the LED lights up when PH.6 and PH.7 are 0 and remains off when they are 1.

To use GPIO, it is necessary to configure specific registers. The memory locations and configuration methods for GPIO settings can be obtained by consulting the manual, as shown in the diagram below.

6.6.6 Register Map

R: read only, **W**: write only, **R/W**: both read and write

Register	Offset	R/W	Description	Reset Value
GPIO Base Address: GPIO_BA = 0x4000_4000				
PA_MODE	GPIO_BA+0x000	R/W	PA I/O Mode Control	0xXXXX_XXXX
PA_DINOFF	GPIO_BA+0x004	R/W	PA Digital Input Path Disable Control	0x0000_0000
PA_DOUT	GPIO_BA+0x008	R/W	PA Data Output Value	0x0000_FFFF
PA_DATMSK	GPIO_BA+0x00C	R/W	PA Data Output Write Mask	0x0000_0000
PA_PIN	GPIO_BA+0x010	R	PA Pin Value	0x0000_XXXX
PA_DBEN	GPIO_BA+0x014	R/W	PA De-bounce Enable Control Register	0x0000_0000
PA_INTTYPE	GPIO_BA+0x018	R/W	PA Interrupt Trigger Type Control	0x0000_0000
PA_INTEN	GPIO_BA+0x01C	R/W	PA Interrupt Enable Control Register	0x0000_0000
PA_INTSRC	GPIO_BA+0x020	R/W	PA Interrupt Source Flag	0x0000_XXXX
PA_SMTEN	GPIO_BA+0x024	R/W	PA Input Schmitt Trigger Enable Register	0x0000_0000
PA_SLEWCTL	GPIO_BA+0x028	R/W	PA High Slew Rate Control Register	0x0000_0000
PA_PUSEL	GPIO_BA+0x030	R/W	PA Pull-up and Pull-down Selection Register	0x0000_0000
GPIO_DBCTL	GPIO_BA+0x440	R/W	Interrupt De-bounce Control Register	0x0000_0020
PAn_PDIO n=0,1..15	GPIO_BA+0x800+(0x04 * n)	R/W	GPIO PA.n Pin Data Input/Output Register	0x0000_000X
PBn_PDIO n=0,1..15	GPIO_BA+0x840+(0x04 * n)	R/W	GPIO PB.n Pin Data Input/Output Register	0x0000_000X
PCn_PDIO n=0,1..15	GPIO_BA+0x880+(0x04 * n)	R/W	GPIO PC.n Pin Data Input/Output Register	0x0000_000X
PDn_PDIO n=0,1..15	GPIO_BA+0x8C0+(0x04 * n)	R/W	GPIO PD.n Pin Data Input/Output Register	0x0000_000X
PEn_PDIO n=0,1..15	GPIO_BA+0x900+(0x04 * n)	R/W	GPIO PE.n Pin Data Input/Output Register	0x0000_000X
PFn_PDIO n=0,1..15	GPIO_BA+0x940+(0x04 * n)	R/W	GPIO PF.n Pin Data Input/Output Register	0x0000_000X
PGn_PDIO n=0,1..12	GPIO_BA+0x980+(0x04 * n)	R/W	GPIO PG.n Pin Data Input/Output Register	0x0000_000X
PHn_PDIO n=0,1..12	GPIO_BA+0x9C0+(0x04 * n)	R/W	GPIO PH.n Pin Data Input/Output Register	0x0000_000X

GPIO 的 Base Address

register address

register description

Based on the required functionality, we need to query the detailed settings of the registers. For this experiment, we need to use PA_MODE and PH_MODE to set GPIO as INPUT/OUTPUT, and access PAn_PDIO and PHn_PDIO to write to or read from data.

Port A-H I/O Mode Control (Px_MODE)

Register	Offset	R/W	Description	Reset Value
PA_MODE	GPIO_BA+0x000	R/W	PA I/O Mode Control	0xXXXX_XXXX
PB_MODE	GPIO_BA+0x040	R/W	PB I/O Mode Control	0xXXXX_XXXX
PC_MODE	GPIO_BA+0x080	R/W	PC I/O Mode Control	0xXXXX_XXXX
PD_MODE	GPIO_BA+0x0C0	R/W	PD I/O Mode Control	0xXXXX_XXXX
PE_MODE	GPIO_BA+0x100	R/W	PE I/O Mode Control	0xXXXX_XXXX
PF_MODE	GPIO_BA+0x140	R/W	PF I/O Mode Control	0xXXXX_XXXX
PG_MODE	GPIO_BA+0x180	R/W	PG I/O Mode Control	0xXXXX_XXXX
PH_MODE	GPIO_BA+0x1C0	R/W	PH I/O Mode Control	0xXXXX_XXXX

31	30	29	28	27	26	25	24
MODE15		MODE14		MODE13		MODE12	
23	22	21	20	19	18	17	16
MODE11		MODE10		MODE9		MODE8	
15	14	13	12	11	10	9	8
MODE7		MODE6		MODE5		MODE4	
7	6	5	4	3	2	1	0
MODE3		MODE2		MODE1		MODE0	

Bits	Description	
[2n+1:2n] n=0,1..15	MODEn	**Port A-H I/O Pin[n] Mode Control** Determine each I/O mode of Px.n pins. 00 = Px.n is in Input mode. 01 = Px.n is in Push-pull Output mode. 10 = Px.n is in Open-drain Output mode. 11 = Px.n is in Quasi-bidirectional mode. **Note 1:** The initial value of this field is defined by CIOINI (CONFIG0 [10]). If CIOINI is set to 0, the default value is 0xFFFF_FFFF and all pins will be quasi-bidirectional mode after chip powered on. If CIOINI is set to 1, the default value is 0x0000_0000 and all pins will be input mode after chip powered on. **Note 2:** Max. n=15 for port A/B/E/G. Max. n=14 for port C/D. Max. n=11 for port F/H.

According to the description in the diagram, the Px_MODE register is 32 bits long, where bits 0 to 1 represent the mode of Px0, bits 2 to 3 represent the mode of Px1, and so on. When the value is 00, it indicates input mode, and when the value is 01, it indicates output mode. Based on this information, we can set the GPIO required for this experiment.

The value of GPIO is determined by bit 0 of Pxn_PDIO. Below is the description of the Pxn_PDIO register.

GPIO Px.n Pin Data Input/Outut Register (Pxn_PDIO)

Register	Offset	R/W	Description	Reset Value
PAn_PDIO n=0,1..15	GPIO_BA+0x800+(0x04 * n)	R/W	GPIO PA.n Pin Data Input/Output Register	0x0000_000X
PBn_PDIO n=0,1..15	GPIO_BA+0x840+(0x04 * n)	R/W	GPIO PB.n Pin Data Input/Output Register	0x0000_000X
PCn_PDIO n=0,1..15	GPIO_BA+0x880+(0x04 * n)	R/W	GPIO PC.n Pin Data Input/Output Register	0x0000_000X
PDn_PDIO n=0,1..15	GPIO_BA+0x8C0+(0x04 * n)	R/W	GPIO PD.n Pin Data Input/Output Register	0x0000_000X
PEn_PDIO n=0,1..15	GPIO_BA+0x900+(0x04 * n)	R/W	GPIO PE.n Pin Data Input/Output Register	0x0000_000X
PFn_PDIO n=0,1..15	GPIO_BA+0x940+(0x04 * n)	R/W	GPIO PF.n Pin Data Input/Output Register	0x0000_000X
PGn_PDIO n=0,1..12	GPIO_BA+0x980+(0x04 * n)	R/W	GPIO PG.n Pin Data Input/Output Register	0x0000_000X
PHn_PDIO n=0,1..12	GPIO_BA+0x9C0+(0x04 * n)	R/W	GPIO PH.n Pin Data Input/Output Register	0x0000_000X

31	30	29	28	27	26	25	24
Reserved							
23	22	21	20	19	18	17	16
Reserved							
15	14	13	12	11	10	9	8
Reserved							
7	6	5	4	3	2	1	0
Reserved							PDIO

Bits	Description	
[31:1]	**Reserved**	Reserved.
[0]	**PDIO**	**GPIO Px.n Pin Data Input/Output** Writing this bit can control one GPIO pin output value. 0 = Corresponding GPIO pin set to low. 1 = Corresponding GPIO pin set to high. Read this register to get GPIO pin status. For example, writing PA0_PDIO will reflect the written value to bit DOUT (Px_DOUT[0]), reading PA0_PDIO will return the value of PIN (PA_PIN[0]). **Note 1:** The writing operation will not be affected by register DATMSK (Px_DATMSK[n]). **Note 2:** Max. n=15 for port A/B/E/G.

V. GPIO Control Design Code

Example Code:

```
1  #include "M480.h"
2
3  int32_t main(void)
4  {
5
6      // Input Pin set
7      GPIO_SetMode(PA, BIT0, GPIO_MODE_INPUT);  // SW1
8      GPIO_SetMode(PA, BIT1, GPIO_MODE_INPUT);  // SW2
9
10     // Output Pin set
11     GPIO_SetMode(PH, BIT6, GPIO_MODE_OUTPUT);  // LEDR1
12     GPIO_SetMode(PH, BIT7, GPIO_MODE_OUTPUT);  // LEDG1
13
14     while(1) {
15
16       PH6 = (PA0 == 0) ? 0 : 1;
17       PH7 = (PA1 == 0) ? 0 : 1;
18
19     }
20 }
```

Setting GPIO Mode

Repetitive reading of button information and LED display

1. Setting GPIO Mode

```
void GPIO_SetMode(GPIO_T *port, uint32_t u32PinMask, uint32_t u32Mode)
{
    uint32_t i;

    for(i = 0ul; i < GPIO_PIN_MAX; i++)
    {
        if((u32PinMask & (1ul << i))==(1ul << i))
        {
            port->MODE = (port->MODE & ~(0x3ul << (i << 1))) | (u32Mode << (i << 1));
        }
    }
}

#define GPIO_PIN_MAX            16UL /*!< Specify Maximum Pins of Each GPIO Port \hideinitializer */
```

Cyclic checking of the required pin (u32PinMask),where GPIO_PIN_MAX is 16.

When the specified pin is detected, set the GPIO mode (u32Mode) to the corresponding bit.

2. Repetitive reading of button information and LED display

```
113   If GPIO PA.0 pin status is high, then set GPIO PA.0 data output to low.
114 */
115 define GPIO_PIN_DATA(port, pin)     (*((volatile uint32_t *)((GPIO_PIN_DATA_BASE+(0x40*(port))) + ((pin)<<2))))
116 define PA0              GPIO_PIN_DATA(0, 0 ) /*!< Specify PA.0 Pin Data Input/Output \hideinitializer */
117 define PA1              GPIO_PIN_DATA(0, 1 ) /*!< Specify PA.1 Pin Data Input/Output \hideinitializer */
118 define PA2              GPIO_PIN_DATA(0, 2 ) /*!< Specify PA.2 Pin Data Input/Output \hideinitializer */
119 define PA3              GPIO_PIN_DATA(0, 3 ) /*!< Specify PA.3 Pin Data Input/Output \hideinitializer */
120 define PA4              GPIO_PIN_DATA(0, 4 ) /*!< Specify PA.4 Pin Data Input/Output \hideinitializer */
121 define PA5              GPIO_PIN_DATA(0, 5 ) /*!< Specify PA.5 Pin Data Input/Output \hideinitializer */
122 define PA6              GPIO_PIN_DATA(0, 6 ) /*!< Specify PA.6 Pin Data Input/Output \hideinitializer */
123 define PA7              GPIO_PIN_DATA(0, 7 ) /*!< Specify PA.7 Pin Data Input/Output \hideinitializer */
124 define PA8              GPIO_PIN_DATA(0, 8 ) /*!< Specify PA.8 Pin Data Input/Output \hideinitializer */
125 define PA9              GPIO_PIN_DATA(0, 9 ) /*!< Specify PA.9 Pin Data Input/Output \hideinitializer */
126 define PA10             GPIO_PIN_DATA(0, 10) /*!< Specify PA.10 Pin Data Input/Output \hideinitializer */
```

The calculation of memory location involves treating the PA0_PDIO register address as the base and adding the offset.

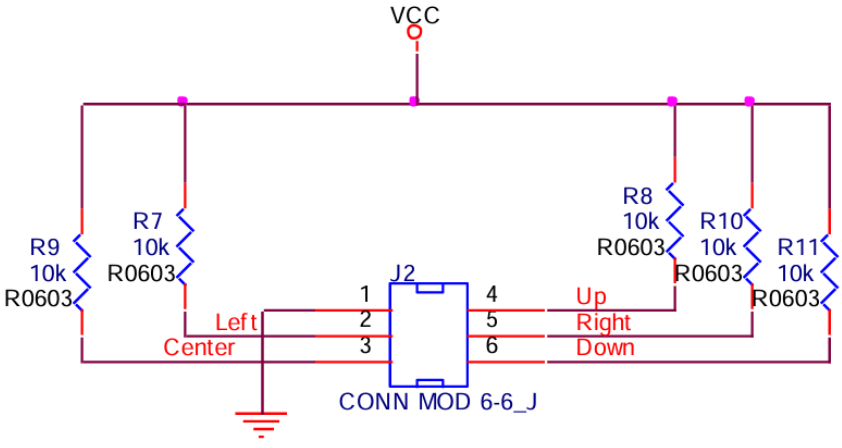
Register	Offset
PAn_PDIO n=0,1..15	GPIO_BA+0x800+(0x04 * n)
PBn_PDIO n=0,1..15	GPIO_BA+0x840+(0x04 * n)
PCn_PDIO n=0,1..15	GPIO_BA+0x880+(0x04 * n)
PDn_PDIO n=0,1..15	GPIO_BA+0x8C0+(0x04 * n)

VI. Implemented

The content implemented in class.

Maintain the original functionality and add LED control using the joystick. When the joystick is pushed up (Up), LEDR1 lights up. When the joystick is pushed down (Down), LEDG1 lights up. When the joystick is pressed (Center), both LEDR1 and LEDG1 light up simultaneously.

The circuit diagram of the joystick is as follows, please refer to page 5 for the control pin assignments.。



Joystick

Experimental Report

The report includes annotated code, program flow, and thoughts.