Lab 1: Interfacing Push-button and LED
Instructor: Prof. Yifeng Zhu
Spring 2016

Goals
1. Get familiar with the Keil uVision software development environment
2. Create a C project for STM32L4 discovery kit and program the kit
3. Learn basics of GPIO input and output configuration
4. Perform simple digital I/O input (interfacing push button) and output (interfacing LED)
5. Understand polling I/O (busy waiting) and its inefficiency

Grading Rubrics (Total = 20 points)
1. Pre-lab assignment (2 points)
2. Documentation and Maintainability (5 points)
3. Functionality and Correctness (5 points)
4. Lab Demonstration (5 points)
5. Something cool (3 points)

NOTE*: Do* ***NOT*** *connect the discovery kit into your PC or laptop before installing the software. Windows might associate the kit with an incorrect USB device driver.*

Pre-lab assignment
1. **Windows** operation system is required. Our development software, Keil uVision, can only run in Windows. If your computer runs Linux or Mac OS, you can install virtual machines.
2. Download free **Keil uVision** (MDK-Lite Edition) and install it. It is free but limits your data and code to 32 KB, which is not an issue for all homework and lab assignments in this course.
3. Follow the tutorial of setting up Gitlab sever. **Set up your public and private keys**
4. Read Textbook **Chapter 4.6** to review bitwise operations.
5. Read Textbook **Chapter 14 GPIO.**

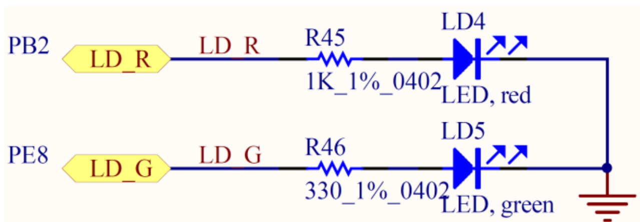
Lab assignment
- Following Book Chapter 14 and implement a C program that toggles both Blue and Green LED when the user button is pressed.
- Do something cool.
  The following gives a few examples but you are not limited to this. ***Creative ideas are always encouraged.***
  - Using an oscilloscope to show the voltage output of LED and the voltage signal of the pin connected to the pushbutton. Find out the latency between the button pressed and LED lighting up.
  - Using the software logic analyzer provided in MDK-KEIL to analyze the digital input and output signals.
  - Using an oscilloscope to show the GPIO output signal difference when the GPIO have different output speeds. Using GPIO a LED to send out SOS in Morse code (· · · – – – · · ·) if the user button is pressed. DOT, DOT, DOT, DASH, DASH, DASH, DOT, DOT, DOT. DOT is on for ¼ second and DASH is on for ½ second, with ¼ second between these light-ons.

LEDs on the Board

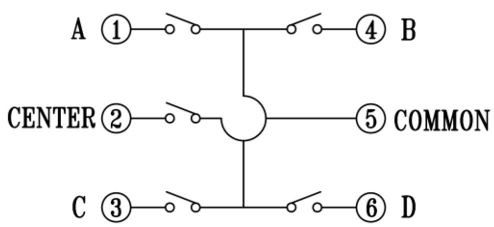
There are two LEDs on the STM32L4 discovery board, which are connected to the GPIO Port B Pin 2 (***PB2***) and the GPIO Port E Pin 8 (***PE8***) pin of the STM32L4 processor, respectively.

To light up a LED, software must at least perform the following three operations:

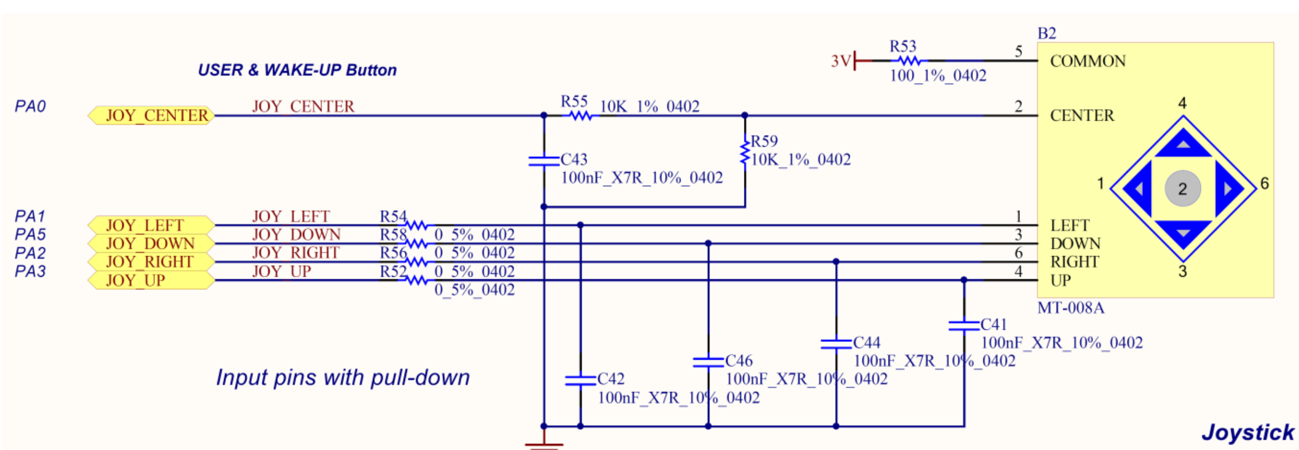
1. Enable the clock of the corresponding GPIO port. (By default, the clock to all peripherals, including GPIO ports, are turned off in order to improve the energy efficiency.)
2. Set the ***mode*** of the corresponding GPIO pin must be set as ***output***, (By default, the mode of all GPIO pin is **analog**)
3. Set the output ***value*** of the corresponding GPIO pin to ***1***. (When the output value is 1, the voltage on the GPIO pin is 3V. When the output value is 0, the voltage is 0V.)



The joystick (MT-008A) has five keys, including *up*, *down*, *left*, *right*, and *select*. Each key has an output pin and all of them are connected to a common pin, as shown below.



The joystick is connected to the GPIO pins PA0, PA1, PA5, PA2, and PA3. A capacitor and a resister are added for each GPIO pin to perform ***hardware debouncing***.



Note: At ambient temperature, the GPIOs (general purpose input/outputs) can sink or source up to **±8 mA**.

PIN Connections

STM32L476VGT6 microcontroller featuring 1 Mbyte of Flash memory, and 128 Kbytes of RAM in LQFP100 package (with 100 pins). The onboard peripherals are connected as follow.

Peripheral	**Peripheral's Interface**	**Pin**	**Peripheral**	**Peripheral's Interface**	**Pin**
Joystick (MT-008A)	Center	PA0	LCD	VLCD	PC3
	Left	PA1		COM0	PA8
	Right	PA2		COM1	PA9
	Up	PA3		COM2	PA10
	Down	PA5		COM3	PB9
User LEDs	LD4 Red	PB2		SEG0	PA7
	LD5 Green	PE8		SEG1	PC5
CS43L22 Audio DAC Stereo I2C address 0x94	SAI1_MCK	PE2		SEG2	PB1
	SAI1_FS	PE4		SEG3	PB13
	SAI1_SCK	PE5		SEG4	PB15
	SAI1_SD	PE6		SEG5	PD9
	I2C1_SCL	PB6		SEG6	PD11
	I2C1_SDA	PB7		SEG7	PD13
	Audio_RST	PE3		SEG8	PD15
MP34DT01 MEMS MIC	Audio_DIN	PE7		SEG9	PC7
	Audio_CLK	PE9		SEG10	PA15
LSM303C eCompass	MAG_CS	PC0		SEG11	PB4
	MAG_INT	PC1		SEG12	PB5
	MAG_DRDY	PC2		SEG13	PC8
	MEMS_SCK	PD1 (SPI2_SCK)		SEG14	PC6
	MEMS_MOSI	PD4 (SPI2_MOSI)		SEG15	PD14
	XL_CS	PE0		SEG16	PD12
	XL_INT	PE1		SEG17	PD10
L3GD20 Gyro	MEMS_SCK	PD1 (SPI2_SCK)		SEG18	PD8
	MEMS_MOSI	PD4 (SPI2_MOSI)		SEG19	PB14
	MEMS_MISO	PD3 (SPI2_MISO)		SEG20	PB12
	GYRO_CS	PD7		SEG21	PB0
	GYRO_INT1	PD2		SEG22	PC4
	GYRO_INT2	PB8		SEG23	PA6
ST-Link V2	USART_TX	PD5	USB OTG	OTG_FS_PowerSwitchOn	PC9
	USART_RX	PD6		OTG_FS_OverCurrent	PC10
	SWDIO	PA13		OTG_FS_VBUS	PC11
	SWCLK	PA14		OTG_FS_ID	PC12
	SWO	PB3		OTG_FS_DM	PA11
	3V3_REG_ON	PB3		OTG_FS_DP	PA12
Quad SPI Flash Memory	QSPI_CLK	PE10(QUADSPI_CLK)	Clock	OSC32_IN	PC14
	QSPI_CS	PE11(QUADSPI_NCS)		OSC32_OUT	PC15
	QSPI_D0	PE12(QUADSPI_BK1_IO0)		OSC_IN	PH0
	QSPI_D1	PE13(QUADSPI_BK1_IO1)		OSC_OUT	PH1
	QSPI_D2	PE14(QUADSPI_BK1_IO2)			
	QSPI_D3	PE15(QUADSPI_BK1_IO3)			

Clock Configuration

There are two major types of clocks: **system clock** and **peripheral clock**.

System Clock

In order to meet the requirement of performance and energy-efficiency for different applications, the processor core can be driven by four different clock sources, including , ***HSI*** (high-speed internal) oscillator clock, ***HSE*** (high-speed external) oscillator clock, **PLL** clock, and **MSI** (multispeed internal) oscillator clock. A faster clock provides better performance but usually consumes more power, which is not appropriate for battery-powered systems.

Peripheral Clock

All peripherals require to be clocked to function. However, ***clocks of all peripherals are turned off by default in order to reduce power consumption***.

The following figure shows the clock tree of ***STM32L476VGT6***, the processor used in the STM32L4 Discovery kit. The clock sources in the domain of Advanced High-performance Bus (***AHB***), low-speed Advanced Peripheral Bus 1 (***APB1***) and high-speed Advanced Peripheral Bus 2 (***APB2***) can be switched on or off independently when it is not used. Software can select various clock sources and scaling factors to achieve desired clock speed, depending on the application's needs.

The software provided in this lab uses the 16MHz HSI as the input to the PLL clock. Appropriate scaling factors have been selected to achieve the maximum allowed clock speed (80 MHz). See the function void **`System_Clock_Init`**() for details.

```
void System_Clock_Init(void){
   ...

  // Enable the Internal High Speed oscillator (HSI)
  RCC->CR |= RCC_CR_HSION;
  while((RCC->CR & RCC_CR_HSIRDY) == 0);

  RCC->CR    &= ~RCC_CR_PLLON;
  while((RCC->CR & RCC_CR_PLLRDY) == RCC_CR_PLLRDY);

  // Select clock source to PLL
  RCC->PLLCFGR &= ~RCC_PLLCFGR_PLLSRC;
  RCC->PLLCFGR |= RCC_PLLCFGR_PLLSRC_HSI; // 00 = No clock, 01 = MSI, 10 = HSI, 11 = HSE

  // Make PLL as 80 MHz
  // f(VCO clock) = f(PLL clock input) * (PLLN / PLLM) = 16MHz * 20/2 = 160 MHz
  // f(PLL_R) = f(VCO clock) / PLLR = 160MHz/2 = 80MHz
  RCC->PLLCFGR = (RCC->PLLCFGR & ~RCC_PLLCFGR_PLLN) | 20U << 8;
  RCC->PLLCFGR = (RCC->PLLCFGR & ~RCC_PLLCFGR_PLLM) | 1U << 4;
  RCC->PLLCFGR &= ~RCC_PLLCFGR_PLLR;  // 00: PLLR = 2, 01: PLLR = 4, 10: PLLR = 6, 11: PLLR = 8
  RCC->PLLCFGR |= RCC_PLLCFGR_PLLREN; // Enable Main PLL PLLCLK output
  RCC->CR   |= RCC_CR_PLLON;
  while((RCC->CR & RCC_CR_PLLRDY) == 0);

  // Select PLL selected as system clock
  RCC->CFGR &= ~RCC_CFGR_SW;
  RCC->CFGR |= RCC_CFGR_SW_PLL; // 00: MSI, 01:HSI, 10: HSE, 11: PLL

  // Wait until System Clock has been selected
  while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_PLL);
```

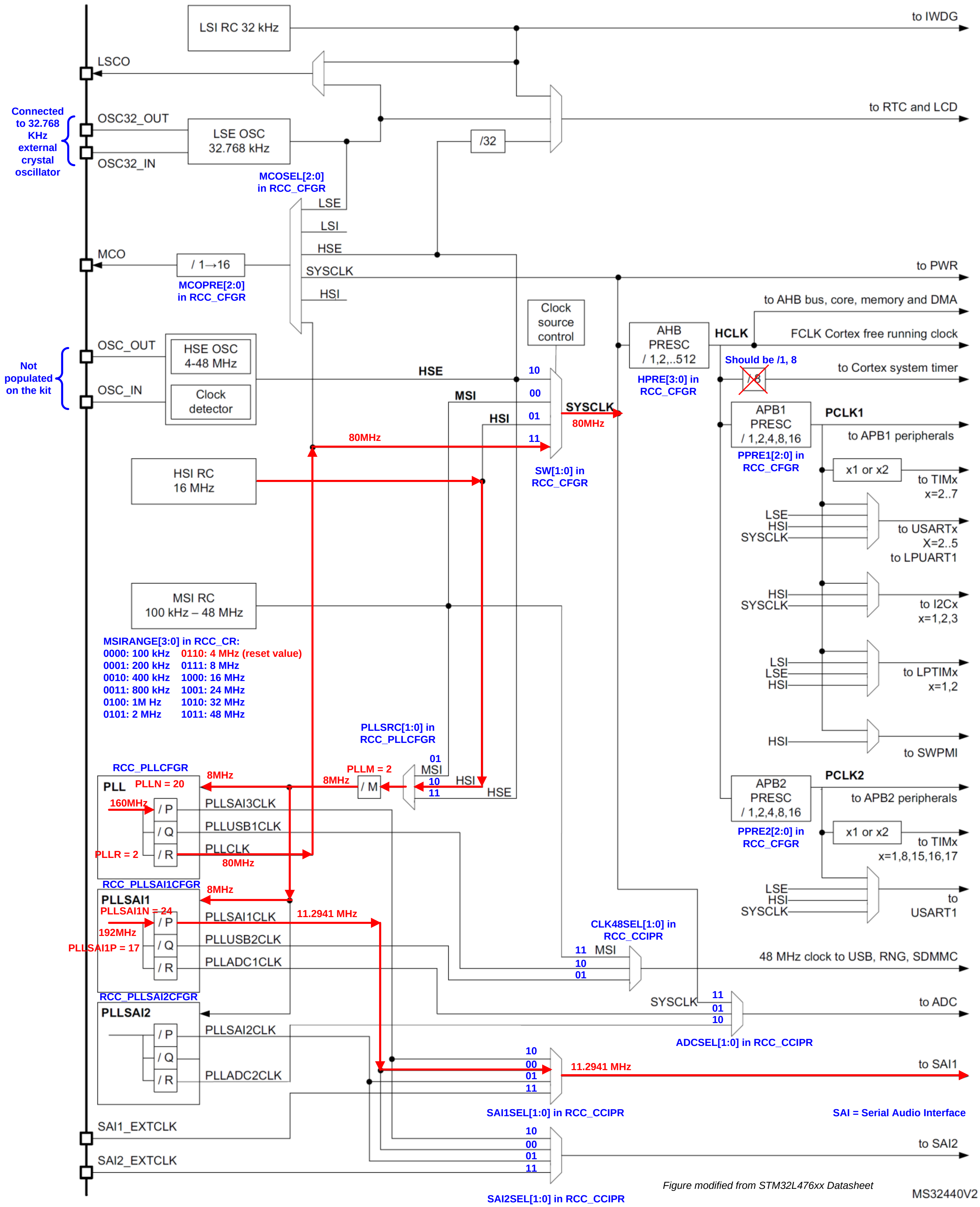
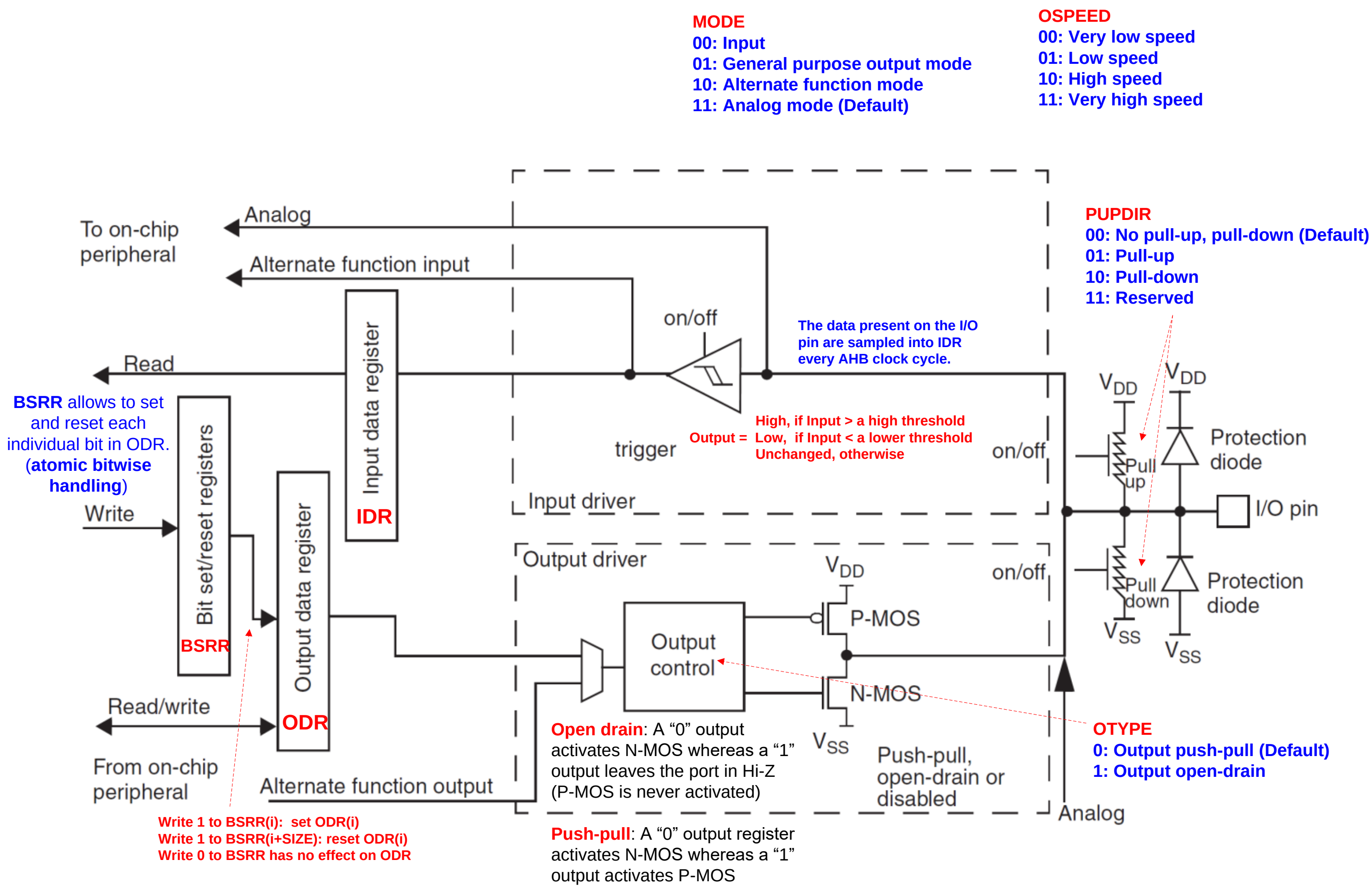


Figure modified from STM32L476xx Datasheet

Each of the GPIO pins can be configured by software as output (push-pull or open-drain), as input (with or without pull-up or pull-down) or as peripheral alternate function. ***In this lab, we will configure PB2 and PE8 as push-pull output.***

Each general-purpose I/O port x (x = A, B, C, …, H) has

- four 32-bit configuration registers
  - GPIOx_MODER (mode register)
  - GPIOx_OTYPER (output type register)
  - GPIOx_OSPEEDR (output speed register)
  - GPIOx_PUPDR (pull-up/pull-down register)
- two 32-bit data registers
  - GPIOx_IDR (input data register)
  - GPIOx_ODR (output data register)
- a 32-bit set/reset register (GPIOx_BSRR).
- a 32-bit locking register (GPIOx_LCKR)
- two 32-bit alternate function selection registers
  - GPIOx_AFRH (alternative function high register)
  - GPIOx_AFRL (alternative function low register)



Code Comments and Documentation

Program comments are used to improve code readability, and to assist in debugging and maintenance. A general principal is "Structure and document your program the way you wish other programmers would" (McCann, 1997).

The book titled "The Elements of Programming Style" by Brian Kernighan and P. J. Plauger gives good advices for beginners.

1. **Format your code well.** Make sure it's easy to read and understand. Comment where needed but don't comment obvious things it makes the code harder to read. If editing someone else's code, format consistently with the original author.

2. Every program you write that you intend to keep around for more than a couple of hours ought to have documentation in it. Don't talk yourself into putting off the documentation. A program that is perfectly clear today is clear only because you just wrote it. Put it away for a few months, and it will most likely take you a while to figure out what it does and how it does it. If it takes you a while to figure it out, how long would it take someone else to figure it out?

3. **Write Clearly** - don't be too clever - don't sacrifice clarity for efficiency.
4. **Don’t over comment.** Use comments only when necessary.
5. Format a program to help the reader understand it. **Always Beautify Code.**
6. Say what you mean, **simply and directly**.
7. **Don't patch bad code** - rewrite it.
8. **Make sure comments and code agree.**
9. Don't just echo code in comments **- make every comment meaningful.**
10. **Don't comment bad code** - rewite it.
11. **The single most important factor in style is consistency.** The eye is drawn to something that "doesn't fit," and these should be reserved for things that are actually different.

Comparison between STM32L1 (used by book) and STM32L4 (used in our lab)

The book was based on STM32L1 board but we are using STM32L4 board this semester. They are slightly different. The following table compares the RCC clock setting and GPIO.

	STM32L1	**STM32L4**
Core	Cortex-M**3** @ 32MHz	Cortex-M**4** @ 80MHz with **FPU and DSP**
MSI	64 kHz, 128 kHz, 256 kHz, 512 kHz, 1.02 MHz, **2.05 MHz (default value)**, 4.1 MHz	100 kHz, 200 kHz, 400 kHz, 800 kHz, 1 MHz, 2 MHz, **4 MHz (default value)**, 8 MHz, 16 MHz, 24 MHz, 32 MHz and 48 MHz
LSI	37 kHz	32 kHz RC
RCC	RCC_AHBENR	RCC_AHB**1**ENR (AHB1) RCC_AHB**2**ENR (AHB2) RCC_AHB**3**ENR (AHB3)
	RCC_AHB**LP**ENR (LP = Low Power)	RCC_AHB**1SM**ENR (AHB1) RCC_AHB**2SM**ENR (AHB2) RCC_AHB**3SM**ENR (AHB3) (SM = Sleep Mode)
	RCC_APB1ENR	RCC_APB1ENR**1** RCC_APB1ENR**2**
	RCC_APB1**LP**ENR (LP = Low Power)	RCC_APB1**SM**ENR**1** RCC_APB1**SM**ENR**2** (SM = Sleep Mode)
	RCC_APB2ENR	RCC_APB2ENR
	RCC_APB2**LP**ENR (LP = Low Power)	RCC_APB2**SM**ENR (SM = Sleep Mode)
GPIO	Default mode is **Digital Input**	Default mode is **Analog**
	Alternative functions are different. AF0 SYSTEM AF1 TIM2 AF2 TIM3/TIM4/TIM5 AF3 TIM9/TIM10/TIM11 AF4 I2C1/I2C2 AF5 SPI1/SPI2 AF6 SPI3 AF7 USART1/ USART2/ USART3 AF8 UART4/UART5 AF9 AF10 USB AF11 LCD AF12 FSMC AF13 AF14 RI AF15 EVENTOUT	Alternative functions are different. AF0 SYSTEM AF1 TIM1/TIM2/TIM5/TIM8/LPTIM1 AF2 TIM1/TIM2/TIM3/TIM4/TIM5 AF3 TIM8 AF4 I2C1/I2C2/I2C3 AF5 SPI1/SPI2 AF6 SPI3/DFSDM AF7 USART1/USART2/ USART3 AF8 UART4/UART5/LPUART1 AF9 CAN1/TSC AF10 OTG_FS/QUADSPI AF11 LCD AF12 SDMMC1/COMP1/COMP2/FMC/SWPMI1 AF13 SAI1/SAI2 AF14 TIM2/TIM15/TIM16/TIM17/LPTIM2 AF15 EVENTOUT
		Add a new register **GPIO_ASCR** (Analog Switch Control Register) 0: Disconnect analog switch to the ADC input

		1: Connect analog switch to the ADC input
		
		`typedef struct {`
		`  __IO uint32_t MODER;`
		`  __IO uint32_t OTYPER;`
		`  __IO uint32_t OSPEEDR;`
		`  __IO uint32_t PUPDR;`
		`  __IO uint32_t IDR;`
		`  __IO uint32_t ODR;`
		`  __IO uint32_t BSRR;`
		`  __IO uint32_t LCKR;`
		`  __IO uint32_t AFR[2];`
		`  __IO uint32_t BRR;`
		`  __IO uint32_t ASCR;`
		`} GPIO_TypeDef;`
		
		For example, to use PA.2 as analog input:
		`GPIOA->ASCR |= 1U<<2;`

Lab 1: Pre-Lab Assignment (2 points)
Spring 2016

Student Name: ______________________

TA: ______________________

Time & Date: ______________________

1. Enable the clock of GPIO Port A (for joy stick), Port B (for Red LED) and Port E (for Green LED)

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AHB2ENR														RNGEN		AESEN			ADCEN	OTGFSEN					GPIOPHEN	GPIOPGEN	GPIOPFEN	GPIOPEEN	GPIOPDEN	GPIOPCEN	GPIOPBEN	GPIOPAEN
Mask																																
Value																																

Note: Why do we need the mask?

When we toggle, set, or reset specific bits of a word (4 bytes), we have to keep the other bits of the word unchanged. For example, we want to set bit 2 of the variable aWord, the following code is incorrect because it resets all the other bits in this word.

```
aWord = 4;
```

The correct approach is:

```
aWord |= 4;
```

Typically we use mask to facilitate the operations of toggling, setting or resetting a group of bits in a variable.

```
Mask = 0x8004;
aWord |= Mask;    // Set bit 15 and bit 2
aWord &= ~Mask;   // Reset bit 15 and bit 2
aWord ^= Mask;    // Toggle bit 15 and bit 2
```

2. Pin Initialization for Red LED (PB 2)

a. Configure PB 2 as Output

GPIO Mode: Input (00), Output (01), Alternative Function (10), Analog (11, default)

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER	MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]		MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
Mask																																
Value																																

GPIOB Mode Register MASK Value = 0x________________________________ (in HEX)

GPIOB Mode Register Value = 0x________________________________ (in HEX)

b. Configure PB 2 Output Type as Push-Pull

Push-Pull (0, reset), Open-Drain (1)

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OTYPER	Reserved																OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
Mask																																
Value																																

GPIOB Output Type Register MASK Value = 0x________________________________ (in HEX)

GPIOB Output Type Register Value = 0x________________________________ (in HEX)

c. Configure PB 2 Output Type as No Pull-up No Pull-down

NO PUPD (00, reset), Pullup (01), Pulldown (10), Reserved (11)

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR	PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]		PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
Mask																																
Value																																

GPIOB Pull-up Pull-down Register MASK Value = 0x________________________________ (in HEX)

GPIOB Pull-up Pull-down Register Value = 0x________________________________ (in HEX)

3. Pin Initialization for Green LED (PE 8)

a. Configure PE 8 as Output

GPIO Mode: Input (00), Output (01), Alternative Function (10), Analog (11, default)

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER	MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]		MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
Mask																																
Value																																

GPIOE Mode Register MASK Value = 0x________________ (in HEX)

GPIOE Mode Register Value = 0x________________ (in HEX)

b. Configure PE 8 Output Type as Push-Pull

Push-Pull (0, reset), Open-Drain (1)

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OTYPER	Reserved																OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
Mask																																
Value																																

GPIOE Output Type Register MASK Value = 0x________________ (in HEX)

GPIOE Output Type Register Value = 0x________________ (in HEX)

c. Configure PE 8 Output Type as No Pull-up No Pull-down

NO PUPD (00, reset), Pullup (01), Pulldown (10), Reserved (11)

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR	PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]		PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
Mask																																
Value																																

GPIOE Pull-up Pull-down Register MASK Value = 0x________________ (in HEX)

GPIOE Pull-up Pull-down Register Value = 0x________________ (in HEX)

4. Pin Initialization for Joy Stick

a. Configure PA0 (Center), PA1 (Left), PA2 (Right), PA3 (Up), and PA5 (Down) as Input

GPIO Mode: Input (00), Output (01), Alternative Function (10), Analog (11, default)

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER	MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]		MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
Mask																																
Value																																

GPIOA Mode Register MASK Value = 0x____________________ (in HEX)
GPIOA Mode Register Value = 0x____________________ (in HEX)

b. Configure PA0 (Center), PA1 (Left), PA2 (Right), PA3 (Up), and PA5 (Down) as Pull-down

NO PUPD (00, reset), Pullup (01), Pulldown (10), Reserved (11)

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR	PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]		PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
Mask																																
Value																																

GPIOE Pull-up Pull-down Register MASK Value = 0x____________________ (in HEX)
GPIOE Pull-up Pull-down Register Value = 0x____________________ (in HEX)

Lab 1: Lab Demo

You should be able to correctly answer the following questions to TAs.

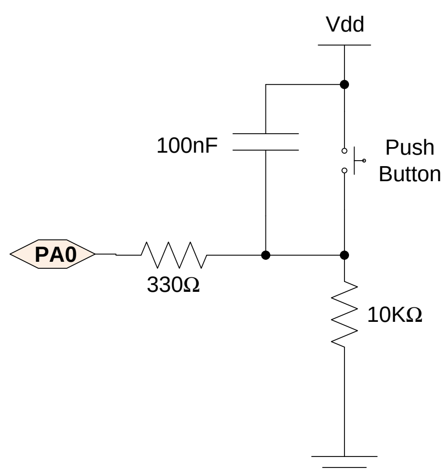
1. Why did we configure the pins that drive the LEDs (PB 2 and PE 8) as push-pull instead of open-drain?
2. What is GPIO speed? What is the default speed? Did you notice any difference of you choose different speeds in this lab assignment?
3. Show TA that you can toggle a LED in the debug environment by directly changing the value of the Output Data Register (ODR) of a GPIO port.

Lab 1: Lab Code Submission

1. Submit and maintain your code in gitlab server
2. Make sure to comment your codes appropriately
3. Make sure to complete the ***Readme.Md*** file

Lab 1: Post-Lab Assignment (1 point)

1. The joy stick on the STM32L4 board has a hardware debouncing circuit. The following is one example debouncing circuit. Explain briefly how the hardware debouncing circuit works. Find out a typical solution for software debouncing.



- Place your answer to this question in the file ***Readme.md***
- Submit it to the Gitlab Server.
- If you have figures/images, you can put your answer in a word document, and put a note in Readme.md. Make sure to perform "***git add***" to the word document.