Lab 1: Interfacing Joystick and LEDs

Instructor: Dr. Yanmin Gong

Teaching Assistants: Francisco E Fernandes Junior and Khuong V. Nguyen

Spring 2019

Goals

1. Get familiar with the System Workbench 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: input/output, push pull, open-drain, pull up/down, GPIO speeds
4. Program GPIO registers to perform simple digital I/O input (interfacing the joystick) and output (interfacing LED)

Grading Rubrics (Total = 100 points)

1. Pre-lab assignment: 10 points.
2. Attendance and Class Participation: 8 points.
3. Code Organization: 8 points.
4. Lab demo questions: 10 points.
5. Primary Objective: 50 points.
6. Secondary Objective: 14 points.

Pre-lab assignment

1. **Download and install the System Workbench for STM32 IDE.** If you need help installing it, consult the **Tutorials** section on D2L.
2. Read Textbook **Chapter 4.6** to review bit-wise operations.
3. Read Textbook **Chapter 14 GPIO.**
4. Watch Youtube Tutorials (http://web.eece.maine.edu/~zhu/book/tutorials.php)
    - Lecture 5: Memory-mapped I/O (8 minutes)
    - Lecture 6: GPIO Output (11 minutes)
    - Lecture 7: GPIO Input (12 minutes)
5. Finish the pre-lab questions. In the pre-lab questions, you are asked what values certain items are in binary and hexadecimal.

Lab Objectives

For this lab, you will have to implement two different functionalities described below.

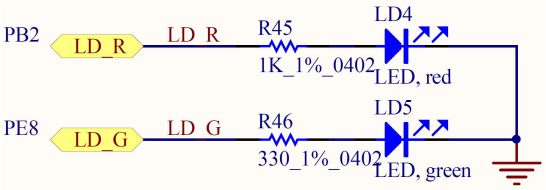
- **Primary Objective (50 points):**
  - Write a **C program** that uses the onboard **joystick** to **control** both the **red** and **green LED**s as follows:
    - **Toggle red LED when the right button is pushed;**
    - **Toggle green LED when the left button is pushed;**
    - **Set both LEDs to on when the up button is pushed;**
    - **Set both LEDs to off when the down button is pushed.**

- **Secondary Objective (14 points):**
  - You must write a program to implement only **ONE** of the following options (choose the one it is the easiest for you):
    - Make the **RED LED** send out **SOS** in **Morse code** (· · · – – – · · ·) if the joystick's middle button is pushed. DOT is on for ¼ second and DASH is on for ½ second, with ¼ second between these light-ons.
    - Write an **Assembly code** that re-implements the **primary functionality**.

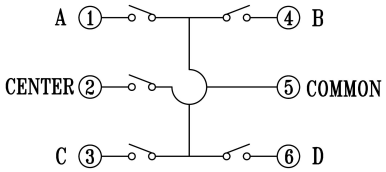
LEDs on the Board

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:

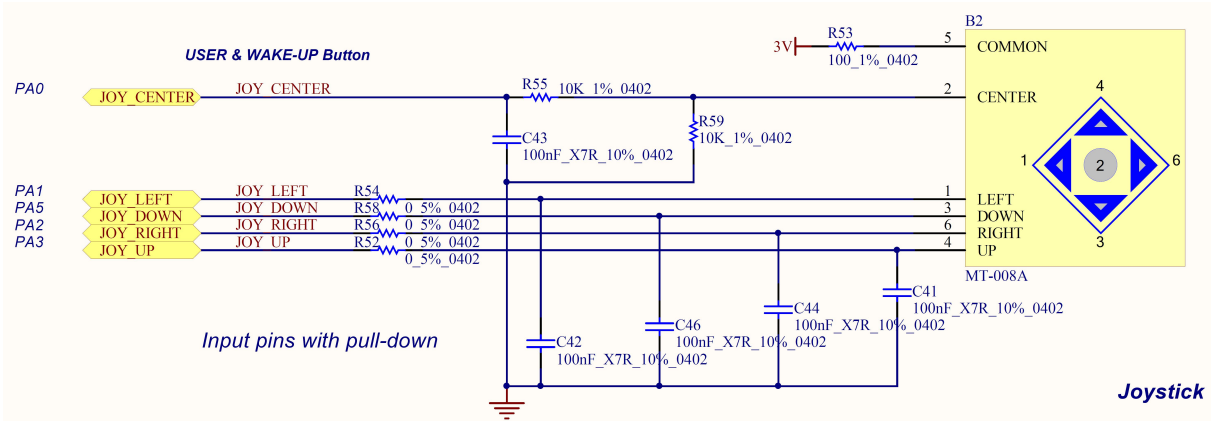
1. Enable the clock of the corresponding GPIO port. (By default, the clock to all peripherals, including GPIO ports, are turned off to improve the energy efficiency.)
2. Set the ***mode*** of the corresponding GPIO pin must be set as ***output*** (By default, the mode of all GPIO pin is **analog**)
3. 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, GPIO pin (general purpose input/outputs) can sink or source up to **±8 mA**.

PIN Connections

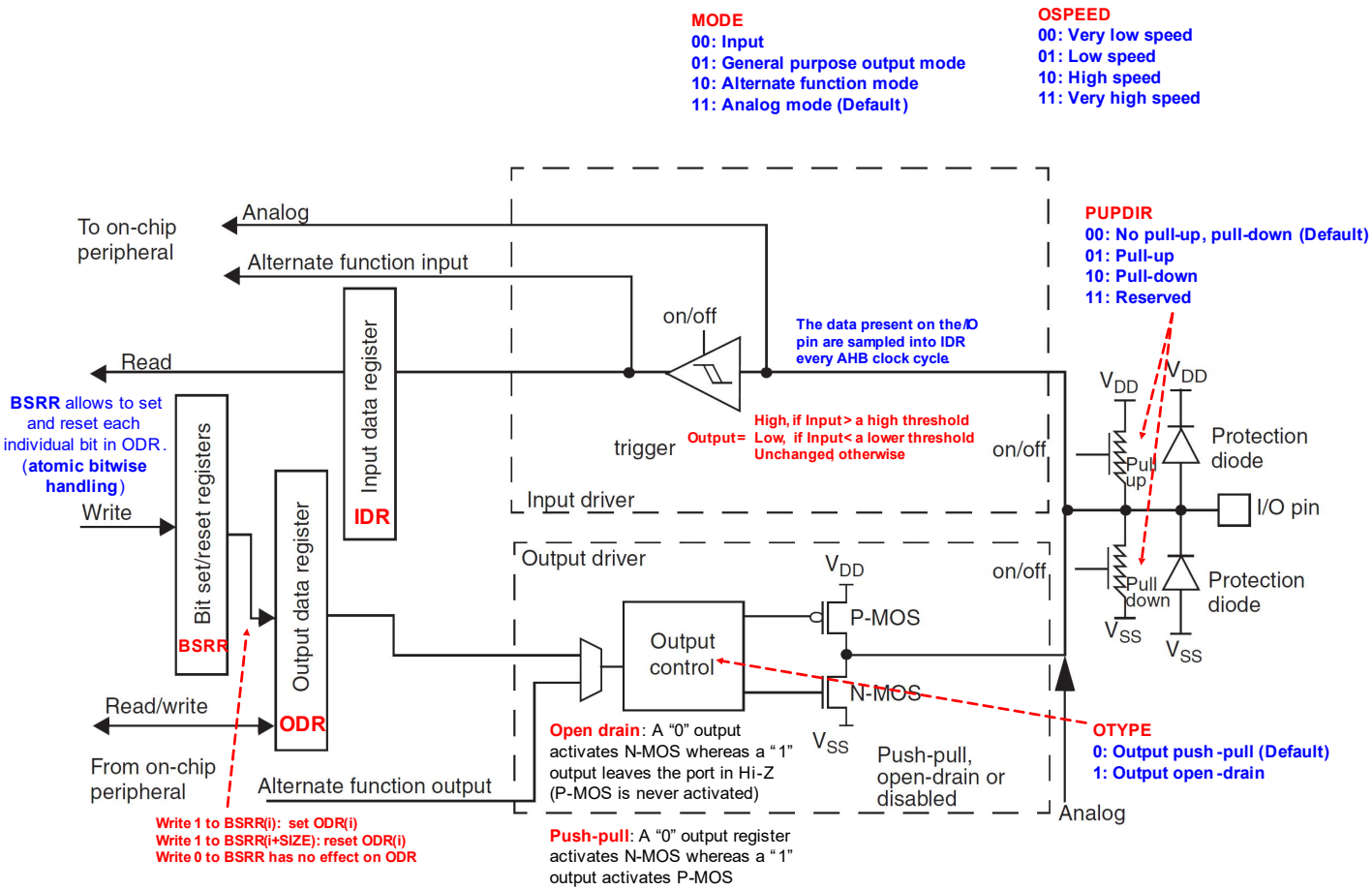
STM32L476VGT6 microcontroller featuring 1 Mbyte of Flash memory, and 128 Kbytes of RAM in LQFP100 package (with 100 pins). The onboard peripherals are connected as follow.

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)			

Introduction to GPIOs

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)


MODE
00: Input
01: General purpose output mode
10: Alternate function mode
11: Analog mode (Default)
OSPEED
00: Very low speed
01: Low speed
10: High speed
11: Very high speed
PUPDIR
00: No pull-up, pull-down (Default)
01: Pull-up
10: Pull-down
11: Reserved
To on-chip peripheral
Analog
Alternate function input
Read
BSRR allows to set and reset each individual bit in ODR. (atomic bitwise handling)
Write
Bit set/reset registers
BSRR
Input data register
IDR
on/off
The data present on the I/O pin are sampled into IDR every AHB clock cycle.
trigger
Output= High, if Input > a high threshold
Low, if Input < a lower threshold
Unchanged otherwise
Input driver
Output driver
Output data register
ODR
Read/write
From on-chip peripheral
Alternate function output
Output control
P-MOS
N-MOS
Open drain: A "0" output activates N-MOS whereas a "1" output leaves the port in Hi-Z (P-MOS is never activated)
Push-pull: A "0" output register activates N-MOS whereas a "1" output activates P-MOS
Push-pull, open-drain or disabled
Write 1 to BSRR(i): set ODR(i)
Write 1 to BSRR(i+SIZE): reset ODR(i)
Write 0 to BSRR has no effect on ODR
Protection diode
Pull up
Pull down
I/O pin
OTYPE
0: Output push-pull (Default)
1: Output open-drain
Analog

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.

Lab 1: Pre-Lab Assignment (10 points)

Mondays students: Due on February 11, 2019 at the beginning of class
Wednesday students: Due on February 13, 2019 at the beginning of class
Print, answer, and hand it back to T.A.
(NO Dropbox submission!)

Student Name: _____

Date: _____

- In this exercise, you should fill the row Mask for a variety of registers. For the tables, always use binary numbers! You should try to write Mask values that when combined with the bitwise set or clear operation gives the desired bit output. You should also write the masks in hexadecimal format.
- This symbol # means a value we don't want to modify, or a value we don't care.
- The initial values of all registers are always considered to be **unknown**!

1. (Example) Enable the clock of GPIO Port A (for joystick), 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					GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
Mask (Set)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1
Desired bit output	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	1	#	#	1	1

RCC->AHB2ENR Register MASK (Bitwise set) Value = 0x**00000013** (in HEX)

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. (2 point) 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 (Clear)																																
Mask (Set)																																
Desired bit output	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	0	1	#	#	#	#

GPIOB Mode Register MASK (Bitwise clear) Value = 0x_____ (in HEX)
GPIOB Mode Register MASK (Bitwise set) Value = 0x_____ (in HEX)

b. (0.5 points) 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 (Clear)																																
Desired bit output																	#	#	#	#	#	#	#	#	#	#	#	#	#	0	#	#

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

c. (1 point) 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 (Clear)																																
Desired bit output	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	0	0	#	#	#	#

GPIOB Pull-up Pull-down Register MASK (Bitwise clear) Value = 0x_____ (in HEX)

3. Pin Initialization for Green LED (PE 8)

a. (2 points) 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 (Clear)																																
Mask (Set)																																
Desired bit output	#	#	#	#	#	#	#	#	#	#	#	#	#	#	0	1	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#

GPIOE Mode Register MASK (Bitwise clear) Value = 0x_____ (in HEX)

GPIOE Mode Register MASK (Bitwise set) Value = 0x_____ (in HEX)

b. (0.5 points) 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 (Clear)																																
Desired bit output																	#	#	#	#	#	#	#	0	#	#	#	#	#	#	#	#

GPIOE Output Type Register MASK (Bitwise clear) Value = 0x_____ (in HEX)

c. (1 point) 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 (Clear)																																
Desired bit output	#	#	#	#	#	#	#	#	#	#	#	#	#	#	0	0	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#

GPIOE Pull-up Pull-down Register MASK (Bitwise clear) Value = 0x_____ (in HEX)

4. Pin Initialization for Joy Stick

a. (1 point) 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 (Clear)																																
Desired bit output	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	0	0	#	#	0	0	0	0	0	0	0	0

GPIOA Mode Register MASK (Bitwise clear) Value = 0x_____ (in HEX)

b. (2 points) 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 (Clear)																																
Mask (Set)																																
Desired bit output	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	1	0	#	#	1	0	1	0	1	0	1	0

GPIOE Pull-up Pull-down Register MASK (Bitwise clear) Value = 0x_____ (in HEX)
GPIOE Pull-up Pull-down Register MASK (Bitwise set) Value = 0x_____ (in HEX)

ENSC 3213 Lab 1 (10 points)
Lab Demo Questions

Student Name: _____

Date: _____

1. Demo your implementation to your lab TA.

2. Write your answer to the following questions (TA will grade them during your lab session).

   - **(5 points)** Why did we configure the pins that drive the LEDs (PB 2 and PE 8) as push-pull instead of open-drain?

   - **(5 points)** What is GPIO output speed? What is the default speed? Did you notice any difference of you choose different speeds in this lab assignment?

Lab 1 – Interfacing Joystick and LEDs
Spring 2019
Grading sheet

This page is the proof of your grade. Keep it until the end of the semester.
Don't forget to ask your TA to fill out this page!

Student name: _____

CWID: _____

Requirements	Your score
Pre-lab assignment (10 points)	
Attendance and Class Participation (8 points)	
Code organization (8 points)	
Lab demo questions (10 points)	
Primary functionality (50 points)	
Secondary functionality (14 points)	
Total:	

Graduate Teaching Assistant:

☐ Francisco E. Fernandes Jr.
☐ Khuong V. Nguyen

_____
TA Signature