Lab 1 - Interfacing Joystick and LED

Graduate Teaching Assistant:

Francisco E. Fernandes Jr.

feferna@okstate.edu

School of Electrical and Computer Engineering

Oklahoma State University

Fall 2018

Objectives

- Get familiar with the STM32L476-Discovery kit and Keil uVision software development environment;
- Light up the blue and green LEDs when the user push a joystick position using C language.

Announcement

- New board:
  - STM32L476 Discovery (ARM Cortex M4 microcontroller)
  - This board is used in the textbook's **third edition**

Announcement

- Dr. Gong will not enforce the use of the textbook's third edition.
- Despite the third edition being based on the new board. There is few changes between first and third editions. The main differences are on the use of register constants and memory positions.
- However, register constants and memory positions can be found on the header file included with the labs (*.h).

```
#define RCC_AHBENR_GPIOBEN (0x00000002)
RCC->AHBENR |= RCC_AHBENR_GPIOBEN;
```

Enable the clock of GPIO port B on the first edition

```
#define RCC_AHB2ENR_GPIOBEN (0x00000002)
RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;
```

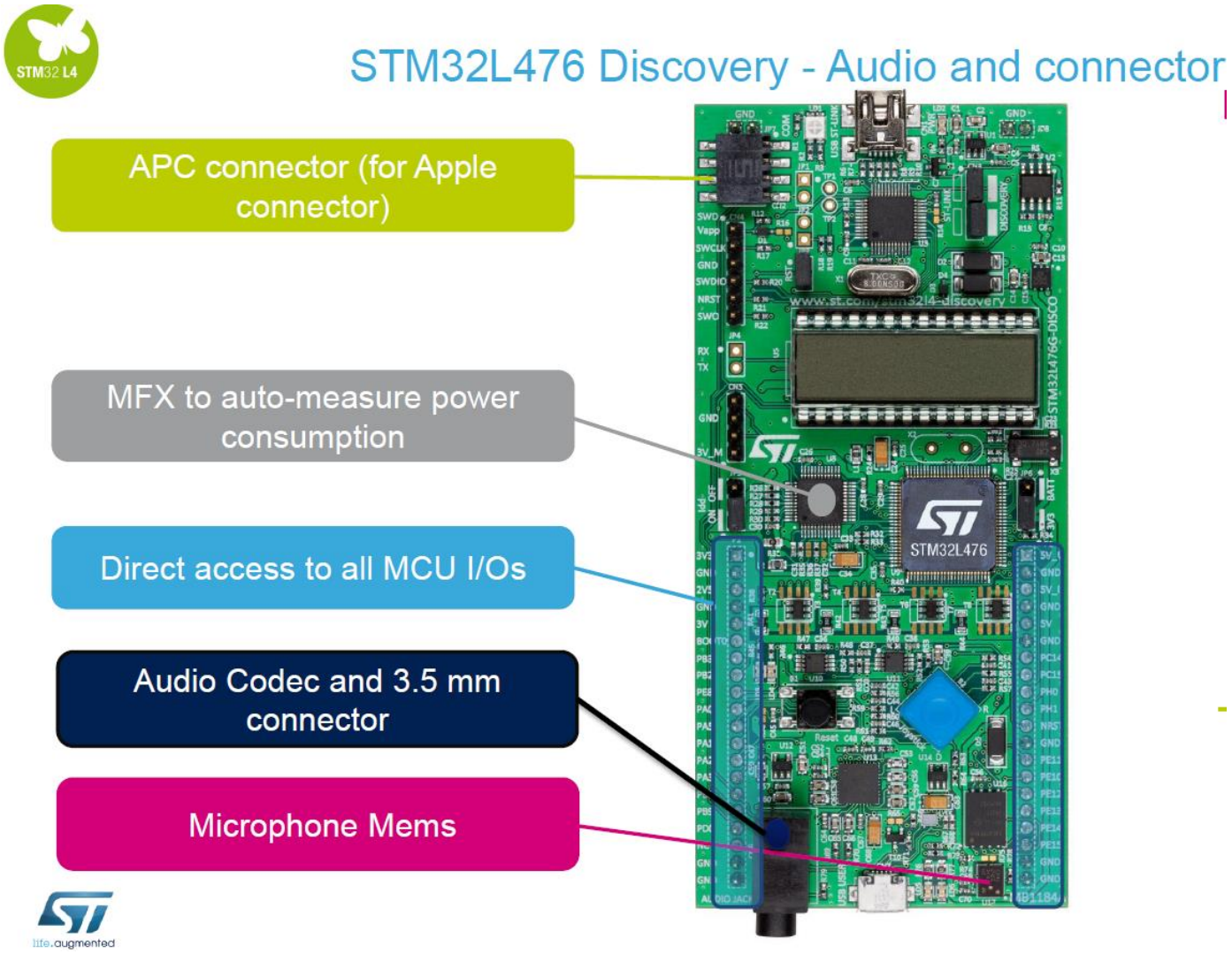
Enable the clock of GPIO port B on the third edition

STML32L476 Discovery

STM32L476 Discovery – HMI

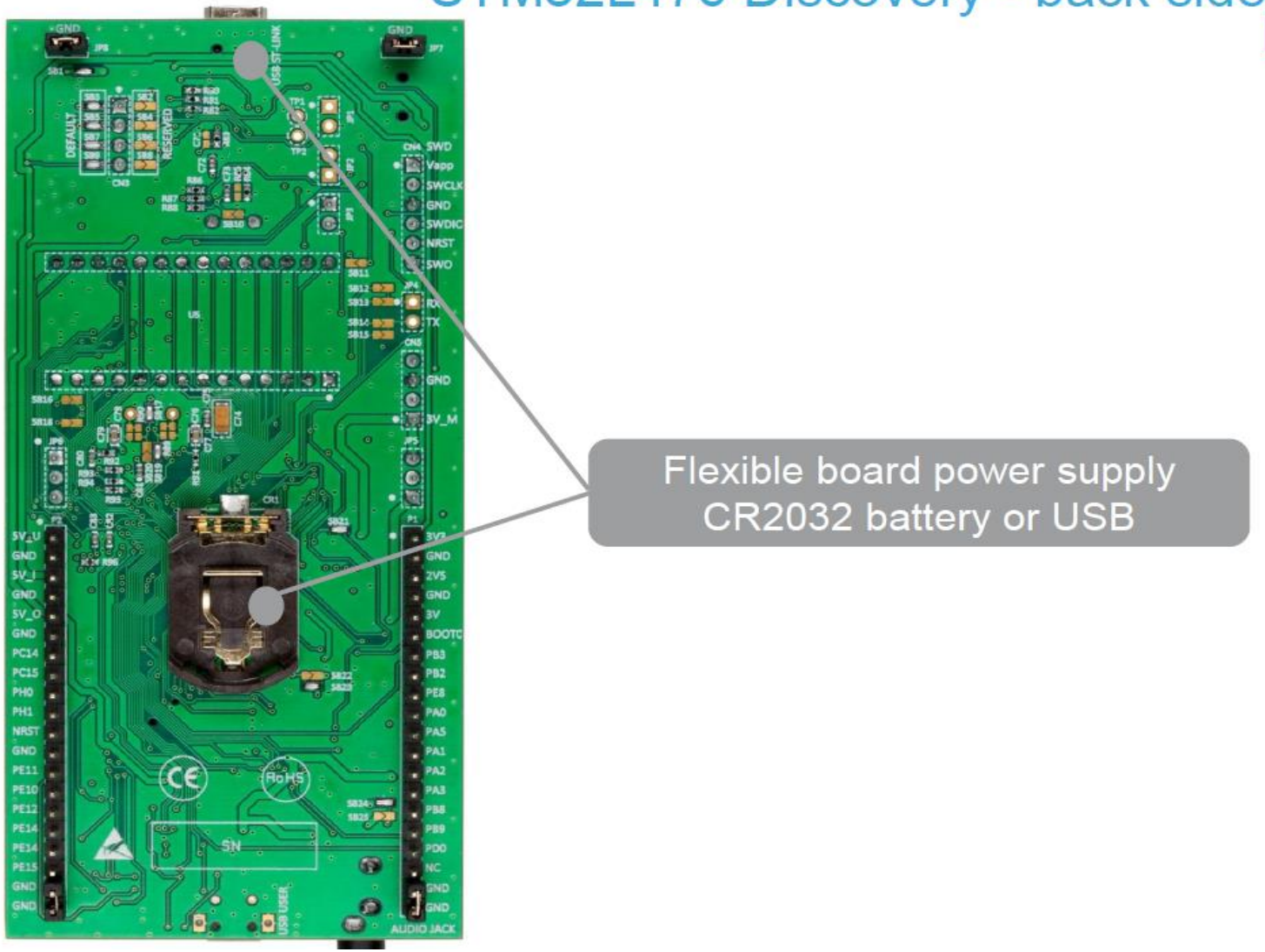
- Integrated ST-Link/V2-1 (for programming and debugging)
- LCD 96 segments
- Motion Mems (9-axis)
- push buttons and joystick, 2 color LEDs
- Quad SPI NOR Flash 16 MB
- USB OTG connector

STML32L476 Discovery



STML32L476 Discovery

STM32L476 Discovery - back side



Bitwise Operations in C

- A = 0xa2; B = 0x34;

AND

A	10100010
B	00110100
A & B	00100000

OR

A	10100010
B	00110100
A | B	10110110

EXCLUSIVE OR

A	10100010
B	00110100
A ^ B	10010110

NOT

A	10100010
~ A	01011101

SHIFT RIGHT

A	10100010
A>>2	00101000

SHIFT LEFT

A	10100010
A<<2	10001000

Bit Operators (&, |, ~) vs Boolean Operators (&& ,||, !)

Don't confuse the bitwise operators & and | with the Boolean (sometimes associated with logical) operators && and ||.

- The Boolean operations are:
  - **A && B** (Boolean and)
  - **A || B** (Boolean or)
  - **!B** (Boolean not)
- The Boolean operations are word-wide operations, not bit-wise operations.
- Example 1:
  - **"0x10 & 0x01"** equals to **0x00**
  - But **"0x10 && 0x01"** equals to **0x01** (Logic True)
- Example 2:
  - **"~0x01"** equals **0xFE**
  - But **"!0x01"** equals to **0x00**

Masking

- With computers, sometimes bits are used to mask bits. That is, they are utilized to turn bits ON or OFF

	A	10100010
	B	11110111
A	B	11110111

- Notice that B is utilized to turn all the bits ON except bit 3, which is kept at its original value.
- Typically, OR is used to turn items ON or **set** a bit and AND is utilized to turn items OFF or **clear** a bit.
- You can also use the original value to turn itself ON or OFF.
- https://en.wikipedia.org/wiki/Mask_(computing)

Check a bit

bit = a & (1<<k)

Example: k = 5

a	a_7	a_6	**a_5**	a_4	a_3	a_2	a_1	a_0
1 << k	0	0	**1**	0	0	0	0	0
a & (1<<k)	0	0	**a_5**	0	0	0	0	0

Set a Bit

a |= (1 << k)

or

a = a | (1 << k)

Example: k = 5

a	a_7	a_6	**a_5**	a_4	a_3	a_2	a_1	a_0
1 << k	0	0	**1**	0	0	0	0	0
a | (1 << k)	a_7	a_6	**1**	a_4	a_3	a_2	a_1	a_0

- In C, operators can be utilized as a shortcut for an operator.
- For example, a += 1 states a = a +1.

Clear a bit

a &= ~(1<<k)

Example: k = 5

a	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
~(1 << k)	1	1	0	1	1	1	1	1
a & ~(1<<k)	a_7	a_6	0	a_4	a_3	a_2	a_1	a_0

Toggle a bit

Without knowing the initial value, a bit can be toggled by XORing it with a "1"

a ^= 1<<k

Example: k = 5

a	a_7	a_6	**a_5**	a_4	A_3	a_2	a_1	a_0
1 << k	0	0	**1**	0	0	0	0	0
a ^= 1<<k	a_7	a_6	NOT(a_5)	A_4	a_3	a_2	a_1	a_0

An exclusive or is useful to see if a bit changes from its previous value, since its 1 iff the value different from its previous value.

a_5	1	$a_5 \oplus 1$
0	1	1
1	1	0

Truth table of Exclusive OR with one

General-Purpose Input and Output (GPIO)

- Each GPIO port has
  - Four 32-bit control registers:
    - GPIO_MODER (digital input, digital output, alternative function, analog input/output)
    - GPIO_OTYPER (output type: push-pull or open-drain)
    - GPIO_OSPEEDR(speed, i.e., slew rate)
    - GPIO_PUPDR (pull-up/pull-down)
  - One 32-bit input data register (GPIO_IDR) and one 32-bit ouput data register (GPIO_ODR):
    - Each bit holds the input/ouput value of one GPIO pin
- Two 32-bit alternative function selection registers (GPIO_AFRH, GPIO_AFRL)
- Clock to GPIO are turned off by default to save power
  - Software program needs to turn on the clock
- The red LED is connected to GPIO port B and green LED is connected to GPIO port E
- The Joystick is connected to GPIO port A

General-Purpose Input and Output (GPIO)

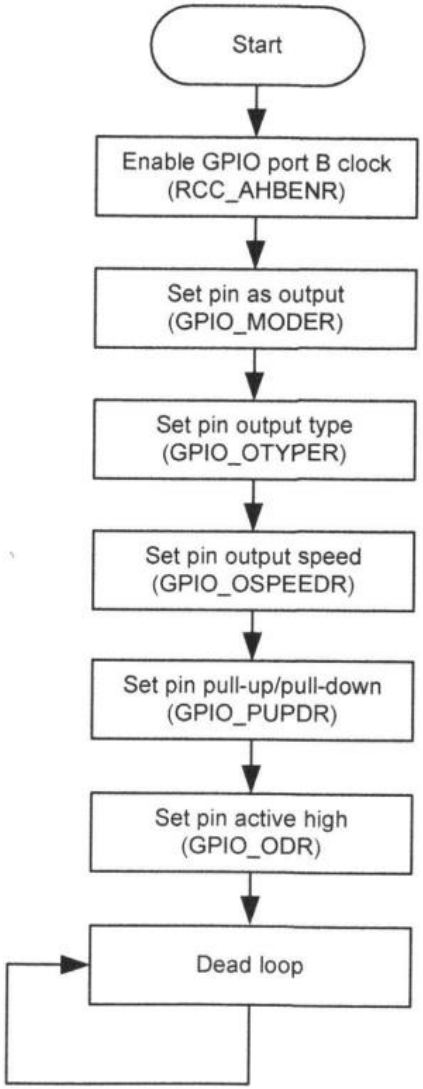


Figure 14-11. Flowchart of GPIO initialization

Lab 1: step-by-step

1. **Enable the clock** to GPIO port A, B and E
2. Configure PB2 (blue LED) and PE8 (green LED) as **output**
3. Configure PB2 and PE8 as **push-pull mode**
4. Configure PB2 and PE8 output type as **No Pull-up No Pull-down**
5. Configure PA3 as **input**
6. Configure PA3 as **No Pull-up No Pull-down**
7. **Wait** until the Joystick UP position is pushed

Thank you!