Introduction to Laboratories

Graduate Teaching Assistants:

Francisco E. Fernandes Jr.

feferna@okstate.edu

Khuong Vinh Nguyen

Khuong.V.Nguyen@okstate.edu

School of Electrical and Computer Engineering

Oklahoma State University

Spring 2019

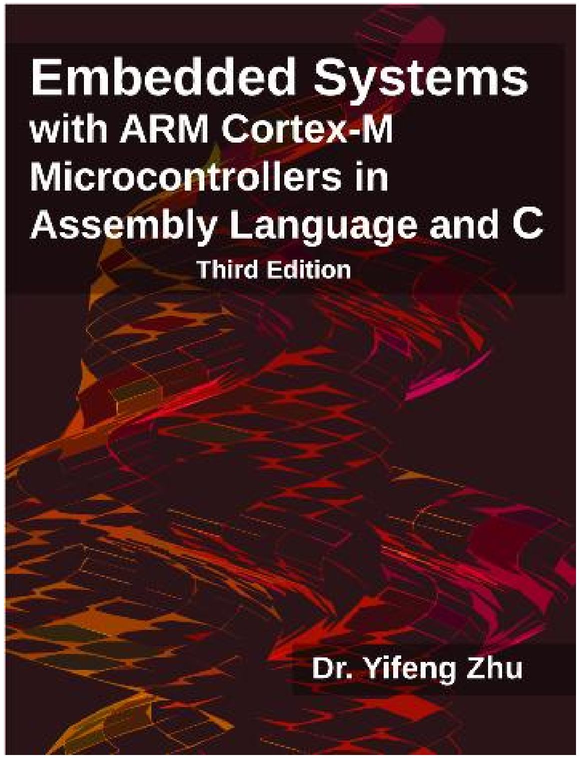

NOTICE
LONG
PANTS
REQUIRED
SmartSign.com • 800-952-1457 • S2-2031

ENDEAVOR General Safety Training Procedure

- Under content on Brightspace, select and watch the **General Endeavor Safety Training** video under ENDEAVOR Safety. You must watch this video for the quiz to become available. The presentation used in the video can be found here also, under **ENDV safety presentation.**
- After you watch the video, go to the Quizzes tab on Brightspace. The **General ENDEAVOR Safety Quiz** should have appeared. You are required to score a 10/10 on this quiz. You have three chances to do so.
- Once you have scored a 10/10 on the quiz, take a screenshot of either of the following screens showing your score and name.
- Print out the screenshot and take it to your TA or instructor. They will confirm that you scored a 10/10 and that the name on the screenshot matches your name.
- After confirming your score and identity, your TA will ask you to sign in, then give you a white ENDEAVOR safety card. You must carry this card with you any time you are in the lab space at ENDEAVOR. ENDEAVOR staff will do random checks.
- For more information: https://ceat.okstate.edu/endeavor/site_files/docs/general_safety_training_procedure.pdf

Required textbook

- Embedded systems with ARM Cortex-M microcontrollers in assembly language and C:
  - **Author**: Dr. Yifeng Zhu
  - **Publisher**: E-Man Press LLC.
  - **ISBN**: 978-0982692660
  - **Third Edition only!**



Optional textbook

- Mastering STM32:
  - **Author**: Carmine Noviello
  - Can be bought in the following link:
    - **https://leanpub.com/mastering-stm32**

What to expect

- **From Dr. Gong's classes:**
  - Fundamental theory about ARM Cortex-M processors.
  - Basics of Assembly programming language.
  - Quizzes and Exams will be in Assembly language.
- **From laboratories classes:**
  - You will learn how to interface hardware and software using an ARM Cortex-M4 development kit.
  - All laboratories are going to be coded in pure C language.
  - Time is limited! You will have to work at home in order to finish all assignments on time!

Labs outline

- **Lab 0:**
  - Introduction to C and ARM programming.
  - 3 weeks
- **Lab 1:**
  - Interfacing a joystick with an LED.
  - 3 weeks.
- **Lab 2:**
  - Stepper Motor Control.
  - 2 weeks.
- **Lab 3:**
  - Liquid Crystal Display (LCD) Driver.
  - 2 weeks.
- **Lab 4:**
  - Interfacing a keypad.
  - 2 weeks.
- **Lab 5:**
  - System Timer.
  - 2 weeks.

Grading Policy

- Your final grade for this course will be calculated as follows:

Final grade breakdown	Percentage
Lectures	54%
Labs	**46%**
Total:	100%

Lab grade breakdown	Percentage
Labs	40%
Midterm exam	2%
Final exam	4%
Total:	**46%**

Grading Policy

Lab 0	
Homework 1	25 points
In-lab assignment 1	25 points
Homework 2	25 points
In-lab assignment 2	25 points
Total:	**100 points**

Labs 1, 2, 3, 4 and 5 (each one)	
Pre-lab assignment	10 points
Attendance and Class Participation	8 points
Code organization	8 points
Lab demo questions	10 points
Primary functionality (basic coding skills)	50 points
Secondary functionality (advanced coding skills)	14 points
Total:	**100 points**

Grading Policy

- Code organization:

```
#include <stdio.h>

int main() {
  printf("Hello World!");
  return 0;
}
```

Good organization!

```
#include <stdio.h>




int main() {  printf("Hello World!");
                    return 0;}
```

Bad organization!

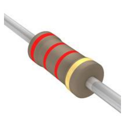
Grading Policy

- **Late submission of assignments will be penalized as follow:**
  - **Up to 24 hours after the deadline:** 50% penalization.
  - **After 24 hours after the deadline:** 100% penalization.
- **All students must work individually! No collaboration is allowed!**
- **Academic Integrity:** The instructor will strictly follow OSU's Academic Integrity Policy. Cheating on homeworks, quizzes or examinations, plagiarism and other forms of academic dishonesty are serious offenses and will subject the student to serious penalties.

Grading Policy

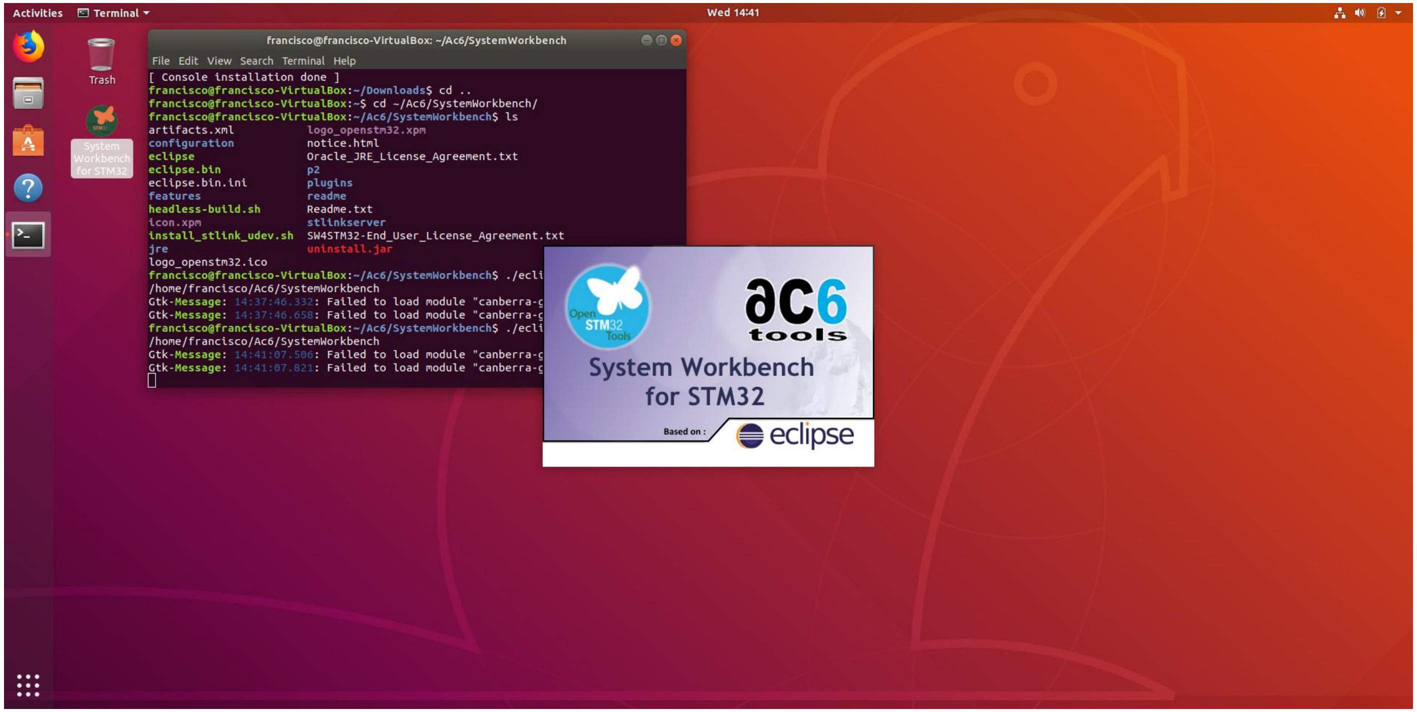
- **Assignments and Exam makeup are only allowed in case of serious illness or accident with the necessary proof!**
  - **It must be requested within 48 hours after the assignment or exam deadline!**
- The following excuses will **NOT** be **accepted** for an assignment or exam makeup:
  - "I forgot the deadline!"
  - "I slept through the exam!"
  - "I missed the bus!"
  - "I didn't know!"
  - "My car broke down!"
  - "I was kicked out from the lab because I didn't come with long pants and shoes!"
  - "I had another exam at the same time!"
  - "I didn't have a printer!"
  - "My computer broke down!"

List of material for labs

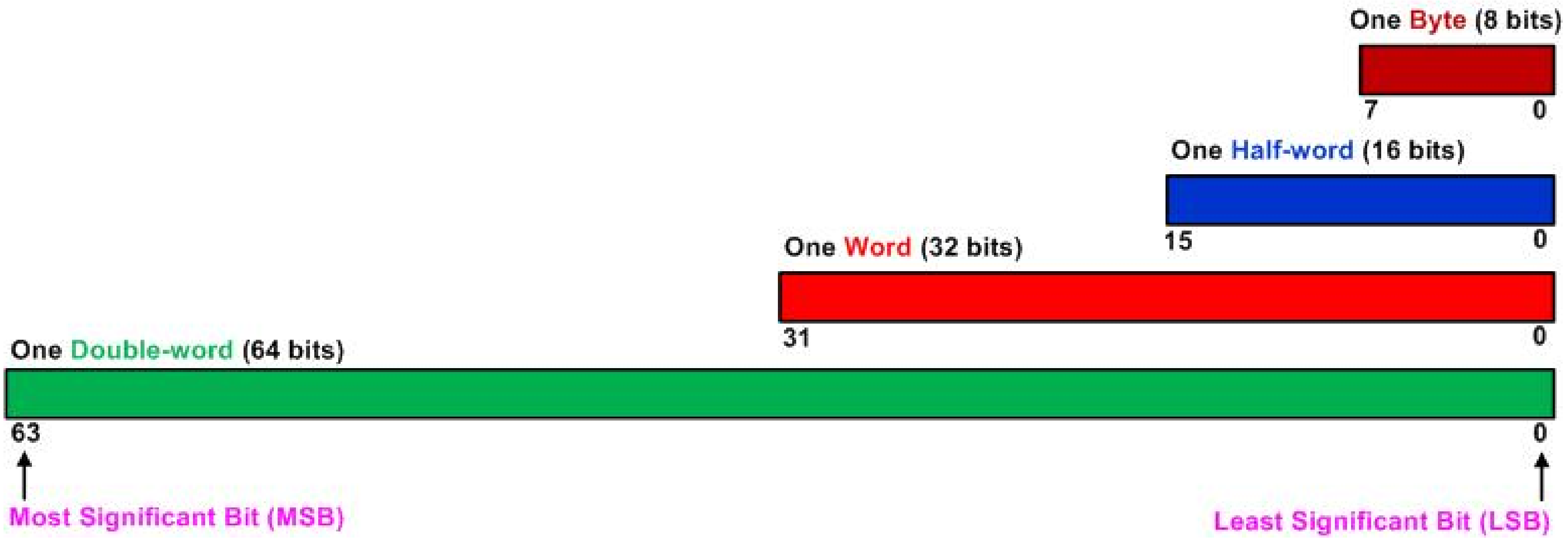
	Description	Where to buy	Price
	Part#: STM32L476G-DISCO	Mouser	$25.00
	One USB cable (A-Male to Mini-B)	Amazon	$4.80
	Two solderless breadboards	Amazon	$9.99
	One 4 x 4 matrix keypad	Amazon	$9.99
	One 28BYJ-48 5v stepper motor + ULN2003 driver board	Amazon	$13.99
	Through hole 2.2 kOhms resistors	Amazon	$5.79
		Total:	**$69.56**

Programming Environment

- **System Workbench for STM32:**
  - Open source GCC-based IDE and compiler.
  - Available for Microsoft Windows, Apple MacOS, and GNU/Linux.
  - http://www.openstm32.org/HomePage



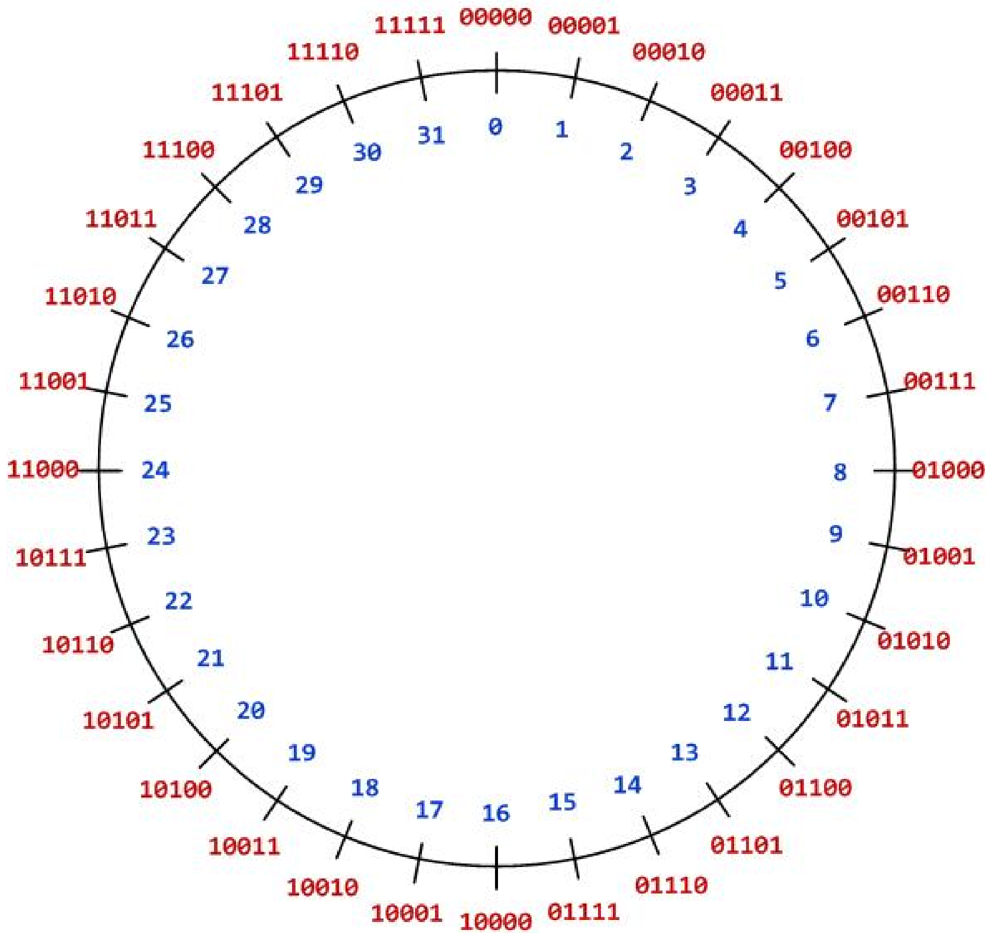
Bit, Byte, Half-word, Word, Double-word



Binary, Octal, Decimal and Hex

Decimal	Binary	Octal	Hex
0	0000	00	0x0
1	0001	01	0x1
2	0010	02	0x2
3	0011	03	0x3
4	0100	04	0x4
5	0101	05	0x5
6	0110	06	0x6
7	0111	07	0x7
8	1000	010	0x8
9	1001	011	0x9
10	1010	012	0xA
11	1011	013	0xB
12	1100	014	0xC
13	1101	015	0xD
14	1110	016	0xE
15	1111	017	0xF

Unsigned Integers



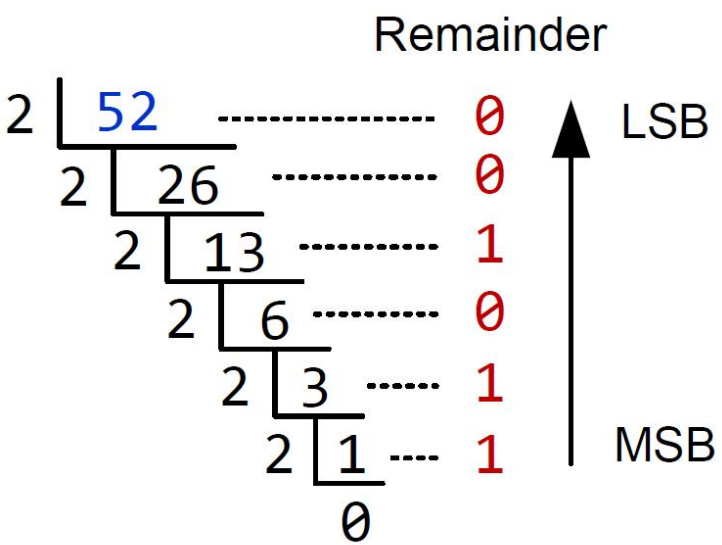
Five-bit binary code

Convert from Binary to Decimal:

$$\mathbf{1011_2}$$
$$= \mathbf{1} \times \mathbf{2^3} + \mathbf{0} \times \mathbf{2^2} + \mathbf{1} \times \mathbf{2^1} + \mathbf{1} \times \mathbf{2^0}$$
$$= \mathbf{8 + 2 + 1}$$
$$= \mathbf{11}$$

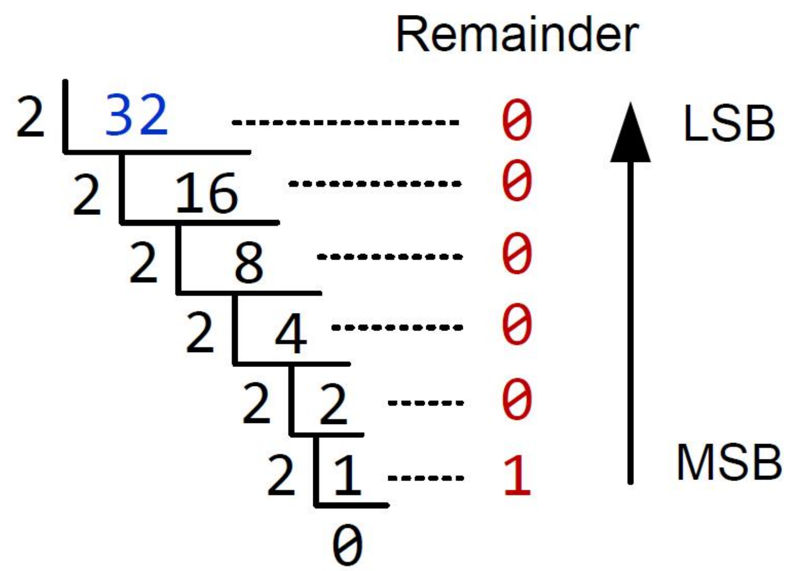
Convert Decimal to Binary

Example 1



$52_{10} = 110100_2$

Example 2

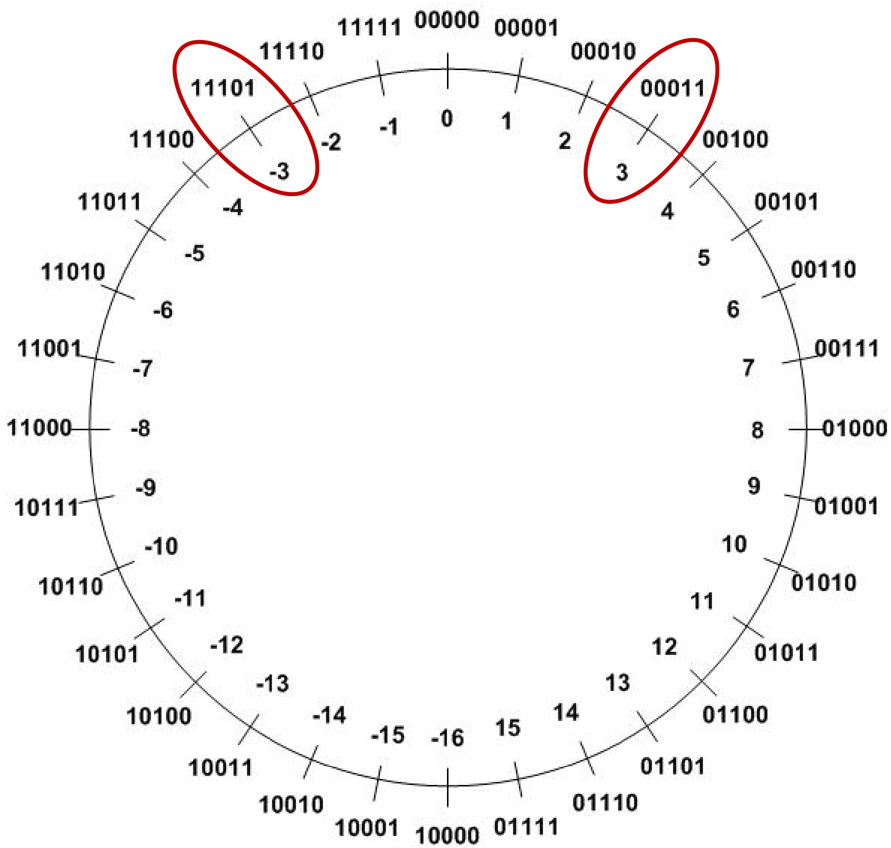


$32_{10} = 100000_2$

Signed Integers: Two's Complement

Two's Complement ($\overline{\alpha}$):

$$\alpha + \overline{\alpha} = 2^n$$



TC of a negative number can be obtained by the bitwise NOT of its positive counterpart plus one.

Example 1:

TC(3)

	Binary	Decimal
Original number	0b00011	3
Step 1: Invert every bit	0b11100	
Step 2: Add 1	+ 0b00001	
Two's complement	0b11101	-3

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 <u>**set**</u> a bit and AND is utilized to turn items OFF or <u>**clear**</u> 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

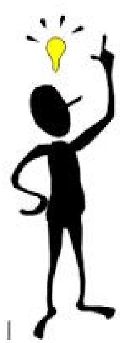
Toggle a bit

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

Examples of masking

- Suppose X is a 8 bit variable with unknown values:
  - X=

X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0

- Answer the following questions:
  - Write a mask, in binary, that makes bits $\mathbf{X_6}$ and $\mathbf{X_2}$ equal to 1 without changing the values of the other bits.

Examples of masking

- Suppose X is a 8 bit variable with unknown values:
  - X=

x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0

- Answer the following questions:
  - Write a mask, in binary, that makes bits $\mathbf{x_6}$ and $\mathbf{x_2}$ equal to 1 without changing the values of the other bits.
  - **Solution:**
    - `Mask = 0 1 0 0 0 1 0 0`

Examples of masking

- Suppose **X** is a 8 bit variable with unknown values:
  - **X**=

x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0

- Answer the following questions:

  - Using the previous **Mask**, what operation should we use to make bits $\mathbf{X_6}$ and $\mathbf{X_2}$ equal to zero?
    a) `X & Mask`
    b) `X | Mask`
    c) `X ^ Mask`
    d) `X & ~(Mask)`

Examples of masking

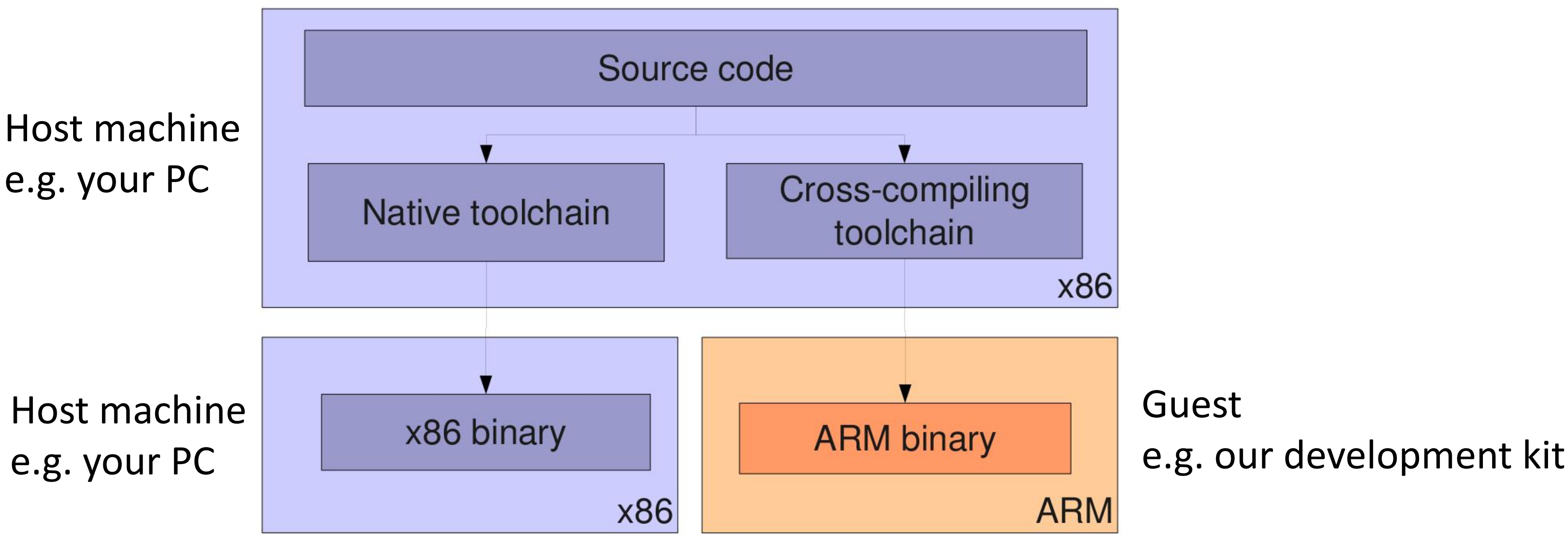
- Suppose **X** is a 8 bit variable with unknown values:
  - **X**=

X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0

- Answer the following questions:
  - Using the previous **Mask**, what operation should we use to make bits $\mathbf{X_6}$ and $\mathbf{X_2}$ equal to zero?
    a) `X & Mask` (Check)
    b) `X | Mask` (Set)
    c) `X ^ Mask` (Toggle)
    d) `X & ~(Mask)` (Clear)

Introduction to C

- The basic requirement of all labs is to write your code in C language.
- However, our C programs will not be the same as the ones used in conventional personal computers.
- We are going to work using **cross-compilation.**



Introduction to C

- Basic structure for ARM processors:

```c
// Always include this header file!
#include "stm32l476xx.h"
// It will help our cross-compiler toolchain
// generate a binary file to be used in our
// development kit.

// Always include a main() function in your code!
// Your code should always contain a file named:
// main.c
int main(void){
  // Configure clock speed

  // Configure peripherals (GPIO)

  // Dead loop & program hangs here
  while(1) {
    // Your program logic goes heres!
  }
}
```

For next class

- **Next class will be on:**
  - **For Mondays class students:** January 28, 2019.
  - **For Wednesdays class students:** January 30, 2019.

- **Homework 1 is due next class!** You can download it online! There will be no Dropbox submission for this assignment!

- **Next subjects:** GPIOs, Register Maps, and more C programming.

Lab 0: Introduction to the Discovery kit and GPIOs

Graduate Teaching Assistants:

Francisco E. Fernandes Jr.

feferna@okstate.edu

Khuong Vinh Nguyen

Khuong.V.Nguyen@okstate.edu

School of Electrical and Computer Engineering

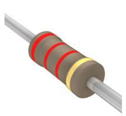
Oklahoma State University

Spring 2019

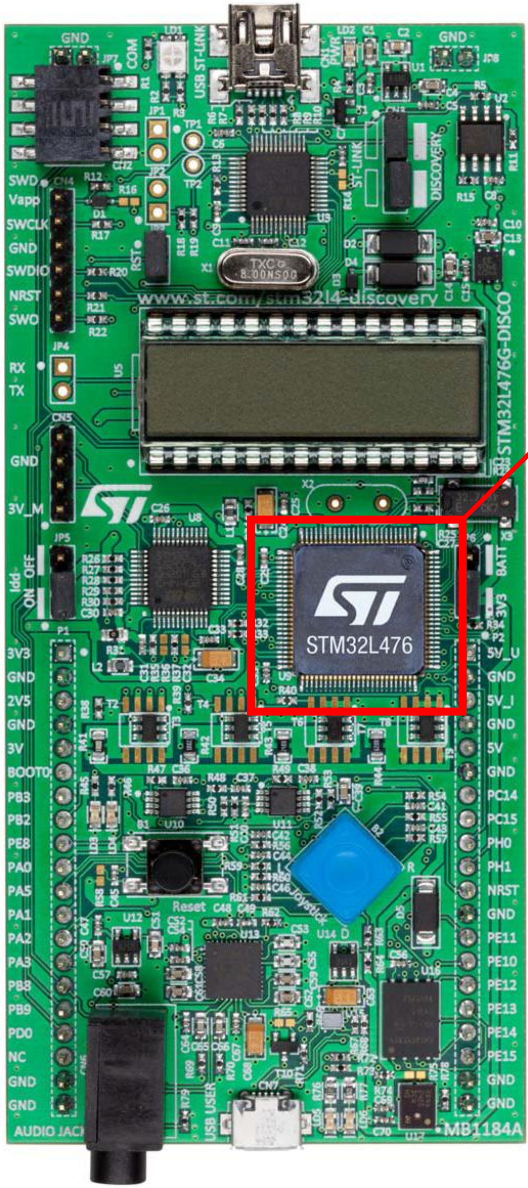
Overview

- **Introduction to STM32L4 Discovery kit.**
  - Firmware programming levels.
    - Hardware Abstraction Layer (HAL).
    - Bare Metal Layer.
  - STM32L4 Clock Structure.
- **General Purpose Input and Output (GPIO):**
  - GPIO operation modes.
  - GPIO registers.
- **In-lab assignment 1.**
- **Homework 2.**

List of material for labs

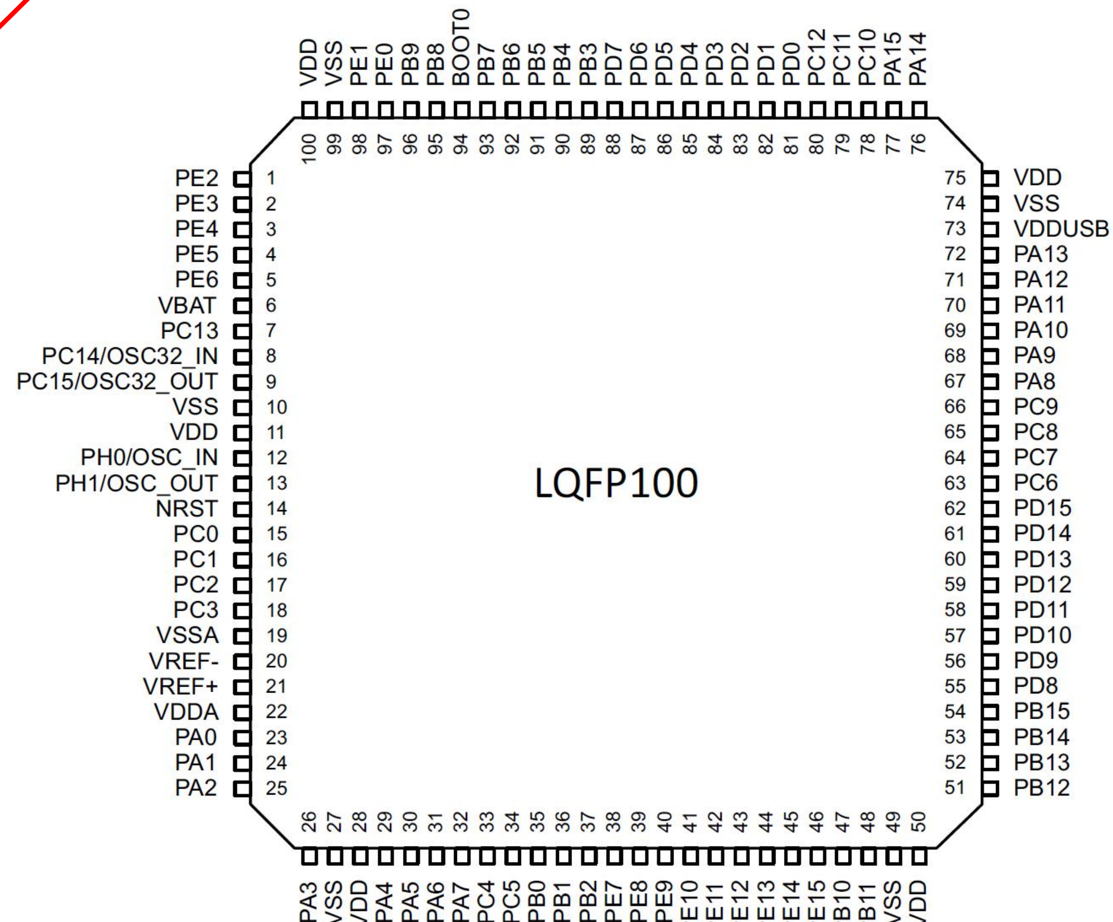
	Description	Where to buy	Price
	Part#: STM32L476G-DISCO	Mouser	$25.00
	One USB cable (A-Male to Mini-B)	Amazon	$4.80
	Two solderless breadboards	Amazon	$9.99
	One 4 x 4 matrix keypad	Amazon	$9.99
	One 28BYJ-48 5v stepper motor + ULN2003 driver board	Amazon	$13.99
	Through hole 2.2 kOhms resistors	Amazon	$5.79
		Total:	**$69.56**

Introduction to the Discovery Kit

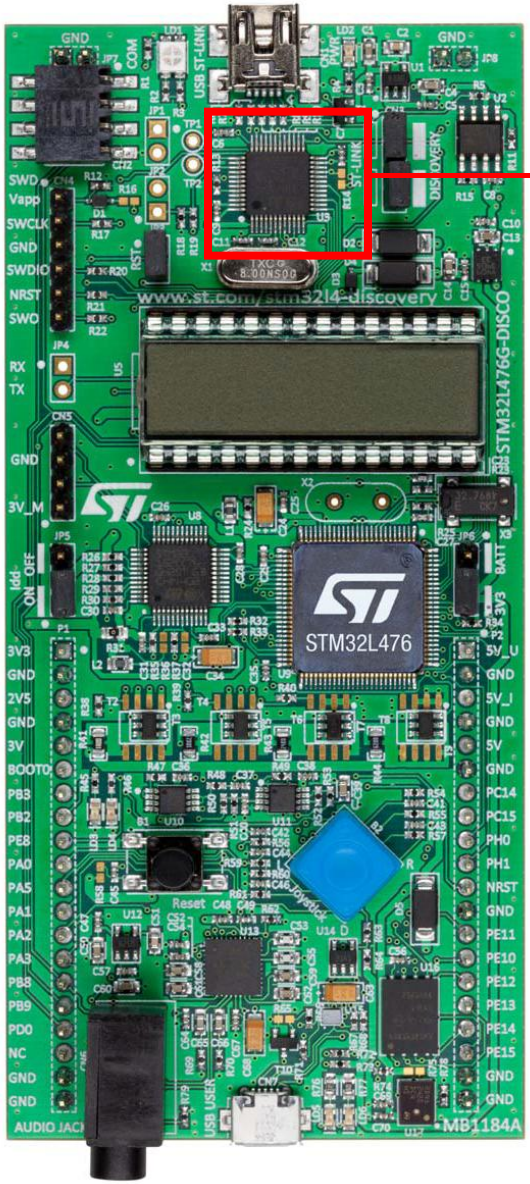


All pins are only 5V tolerant. Do not burn it!

STM32L476G

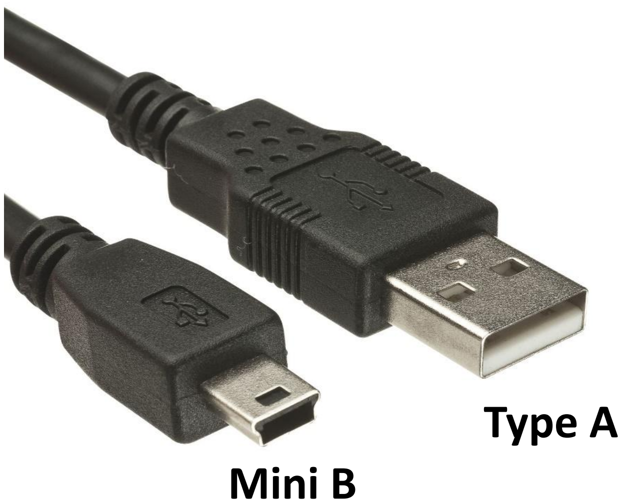


Introduction to the Discovery Kit

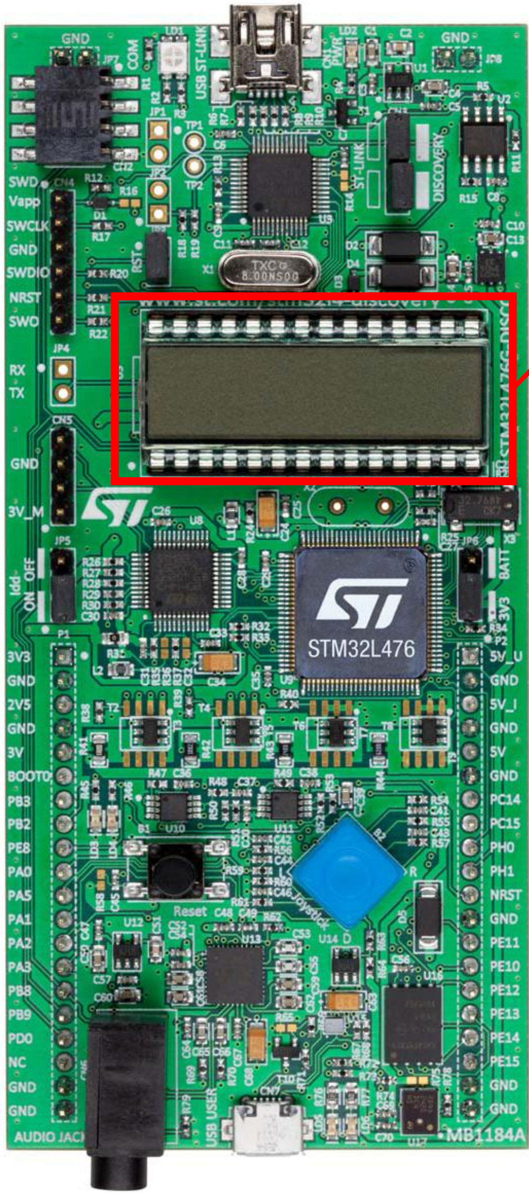


ST-Link / V2-1

- **For programming and debugging**
- **Implemented by using an ARM Cortex-M3**

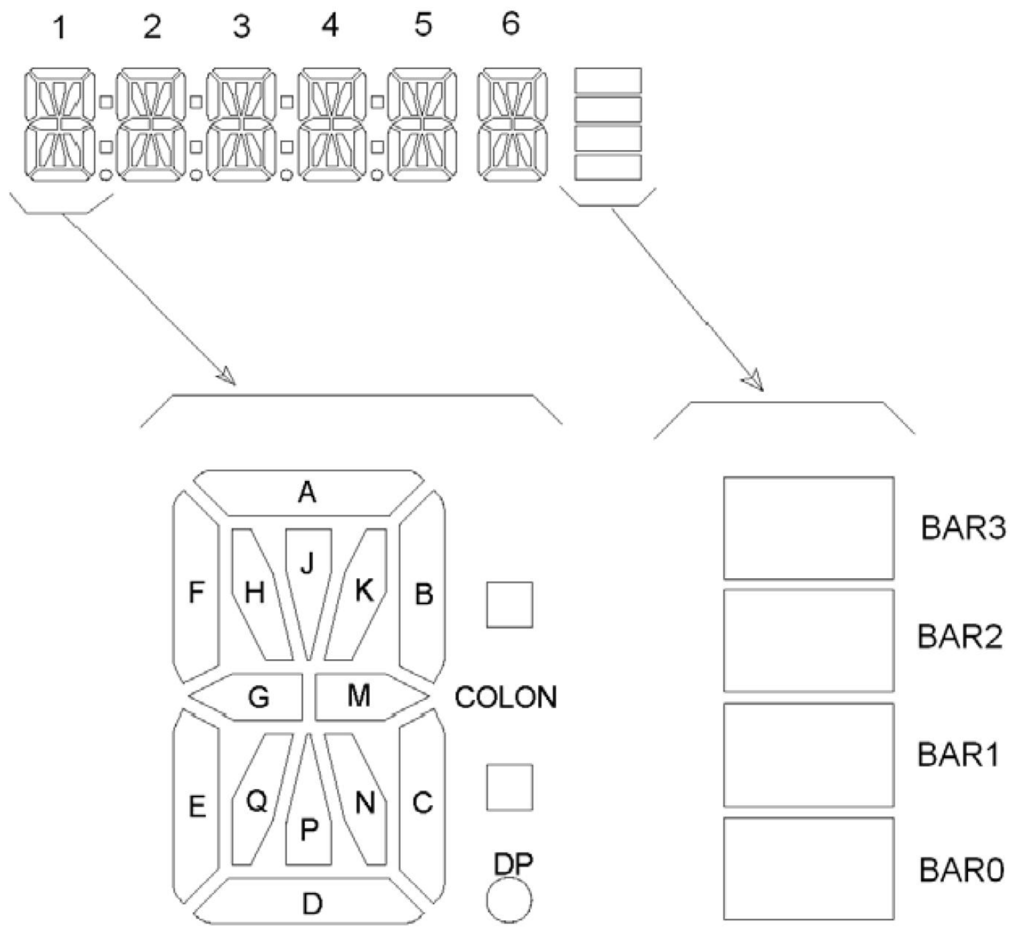


Introduction to the Discovery Kit

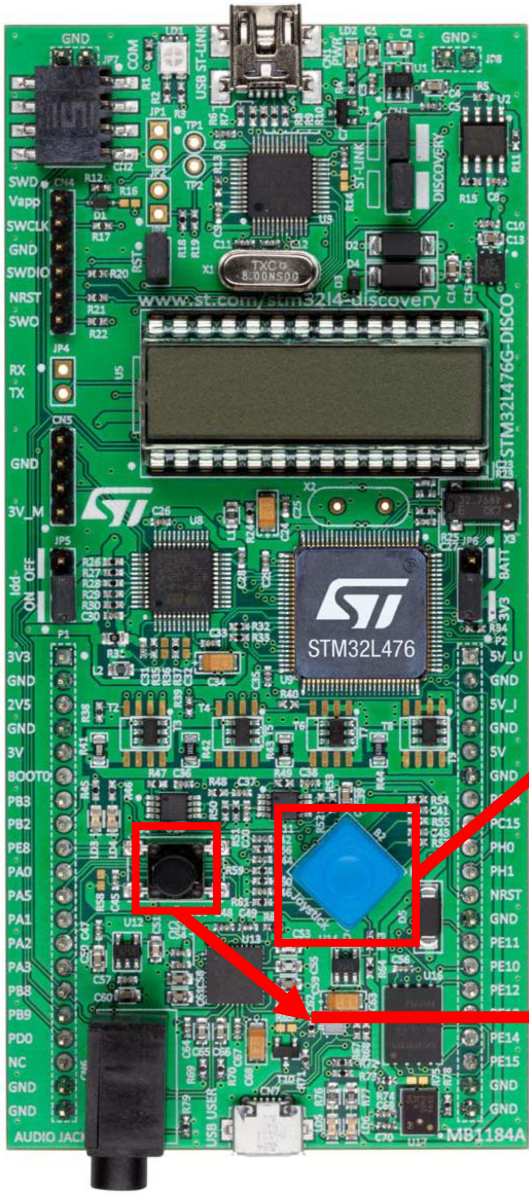


LCD

- **96 segments/pixels**
- **DIP 28 package (24 segments, 4 commons)**



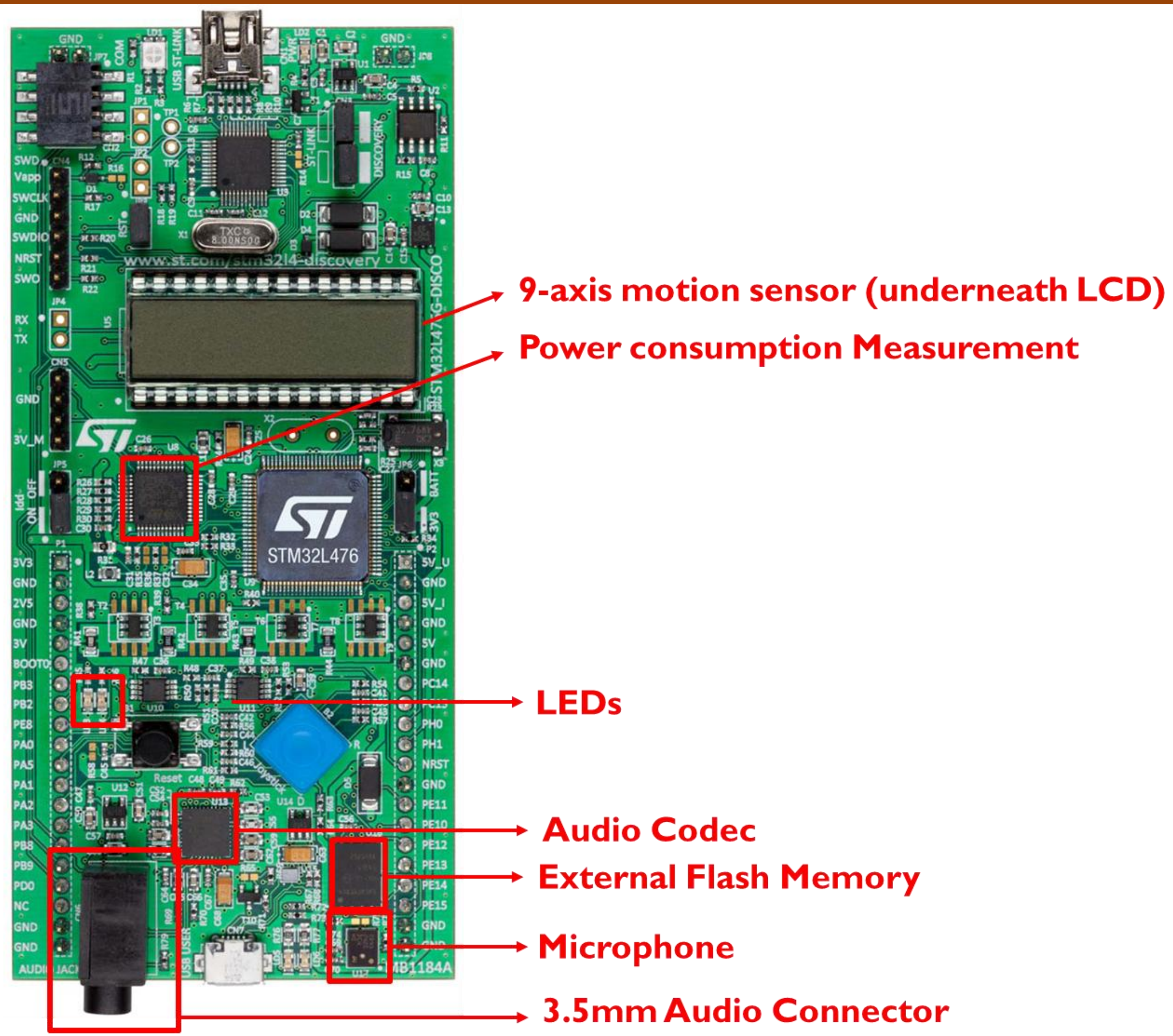
Introduction to the Discovery Kit



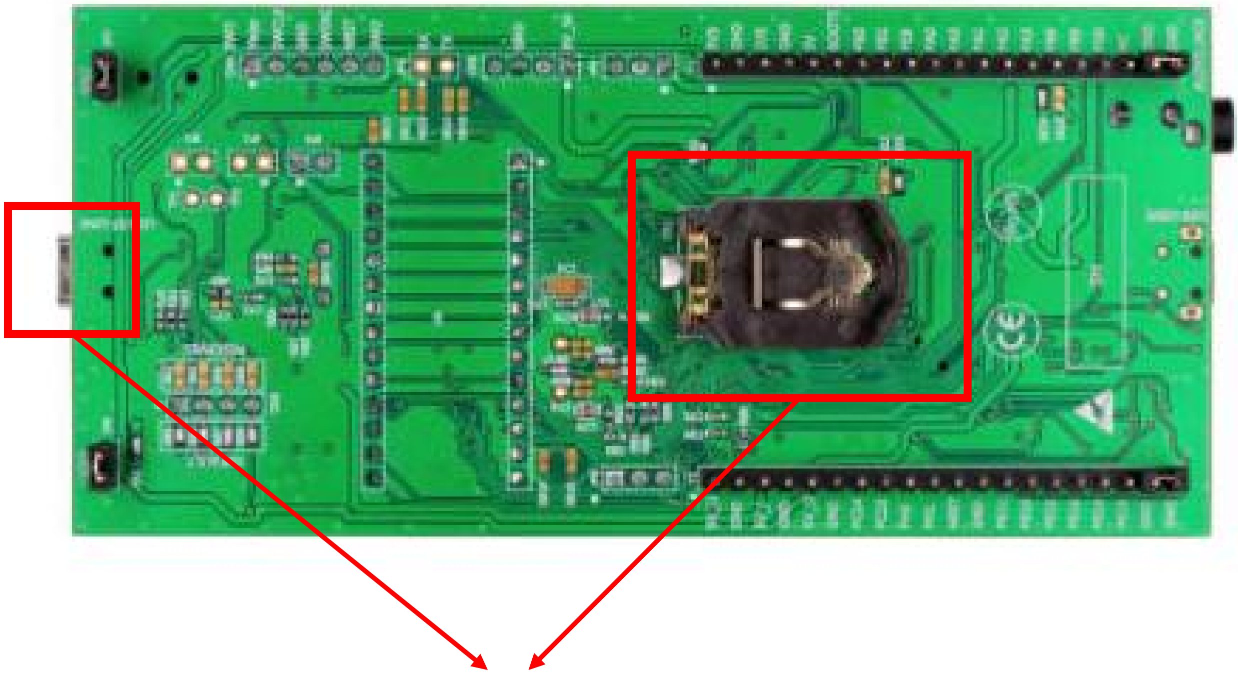
Joystick (up, down, left, right, center)

Pushbutton (reset)

Introduction to the Discovery Kit



Introduction to the Discovery Kit



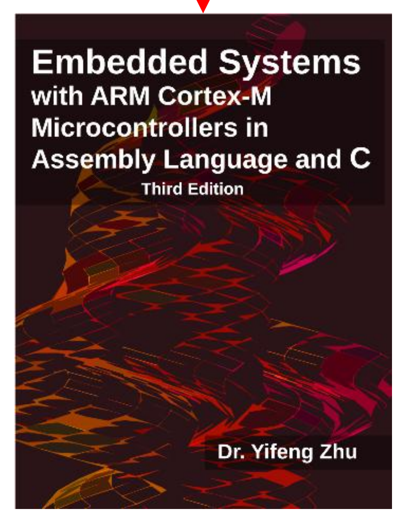
Powered either by

- **USB**
- **3V Coin Battery (CR2032)**

Introduction to the Discovery Kit

- **Software Development:**

	Hardware Abstract Layer	Bare Metal Layer
Library	A rich set of functions	Does not use any library
Productivity	High	Low
Overhead	Typically Low	Zero
Portability	Medium	Difficult
Flexibility	Low	High



Introduction to the Discovery Kit

- **Software Development:**

	Hardware Abstract Layer	Bare Metal Layer
Library	A rich set of functions	Does not use any library
Productivity	High	Low
Overhead	Typically Low	Zero
Portability	Medium	Difficult
Flexibility	Low	High

- **Our labs** will be programmed at the **Bare Metal Layer** level!
- Programming at the bare metal layer requires great knowledge of the target hardware platform.
- Learn how to control or interface a peripheral (such as GPIO, timer, UART, SPI) directly at the register level.
- The book focuses on the programming in the bare metal layer in C and Assembly.

Introduction to the Discovery Kit

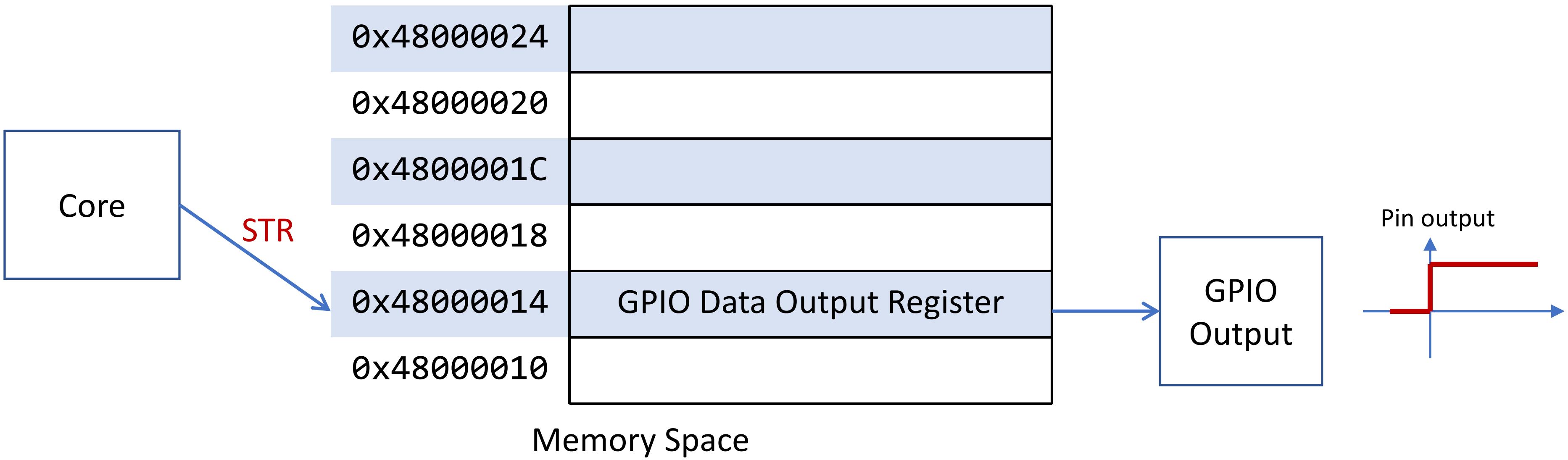
- **Reset and Clock Controller (RCC):**
  - Systems and peripherals can be driven by various clock sources and speeds.
    - To meet the application's requirements on power consumption and accuracy.
  - Peripherals are turned off by default to reduce power consumption.
    - Software have to enable the clock when interacting with a peripheral.
  - RCC module manages the clock and reset of systems and peripherals.

Introduction to the Discovery Kit

- **Clocks:**
  - **Clock source selection:** tradeoff between energy-efficiency, accuracy and performance
    - **Internal clock:** HSI, MSI, LSI (32.768KHz)
    - **External clock:** HSE, LSE
    - **Three PLLs** (Phase-Locked Loop)
  - **Maximum frequency to systems:** 80 MHz

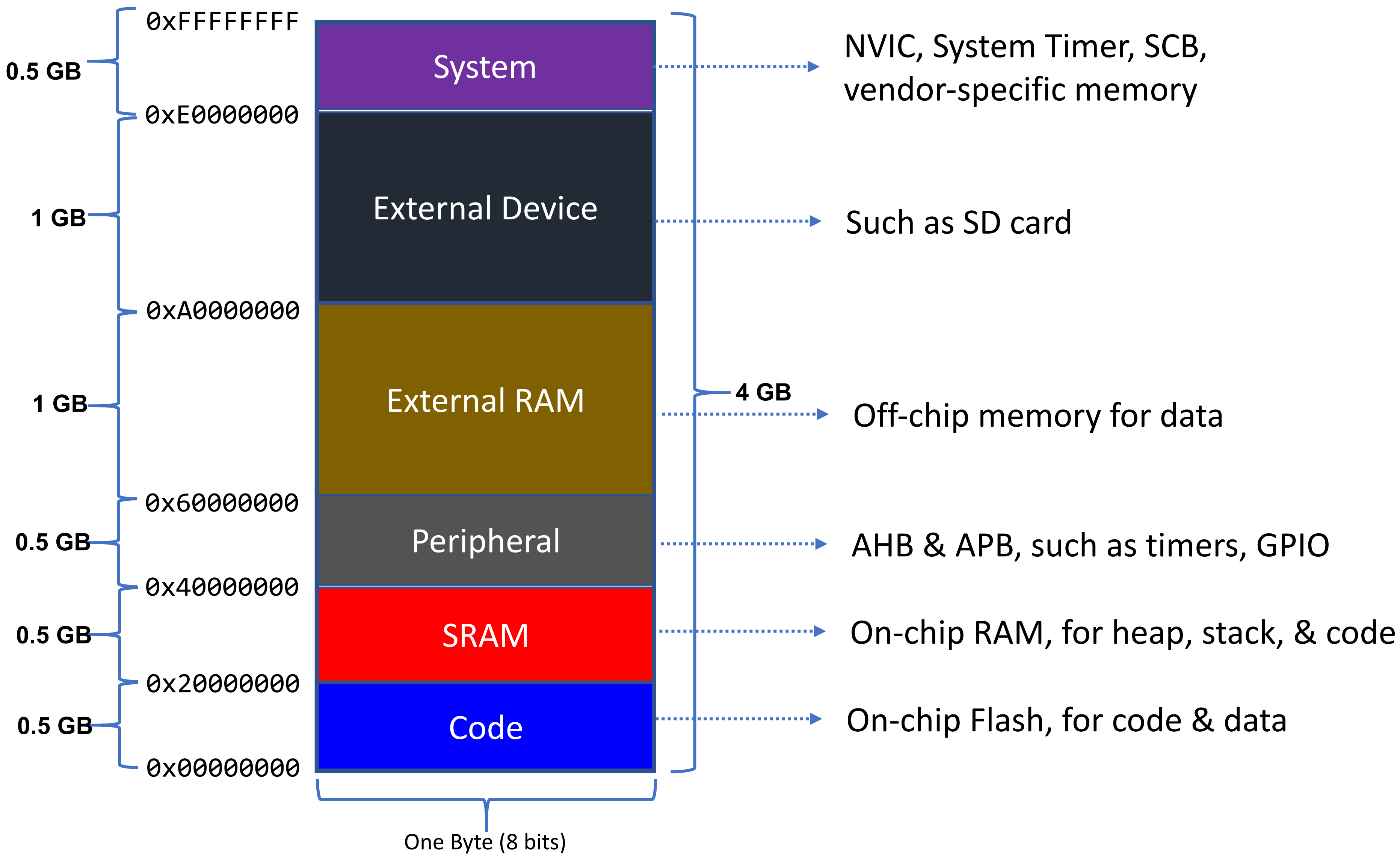
Introduction to GPIOs

- **Interfacing Peripherals:**
  - **Port-mapped I/O**
    - Use special CPU instructions: `Special_instruction Reg, Port`
  - **Memory-mapped I/O**
    - A simpler and more convenient way to interface I/O devices
    - Each device registers is assigned to a memory address in the address space of the microprocessor
    - Use native CPU load/store instructions: `LDR/STR Reg, [Reg, #imm]`

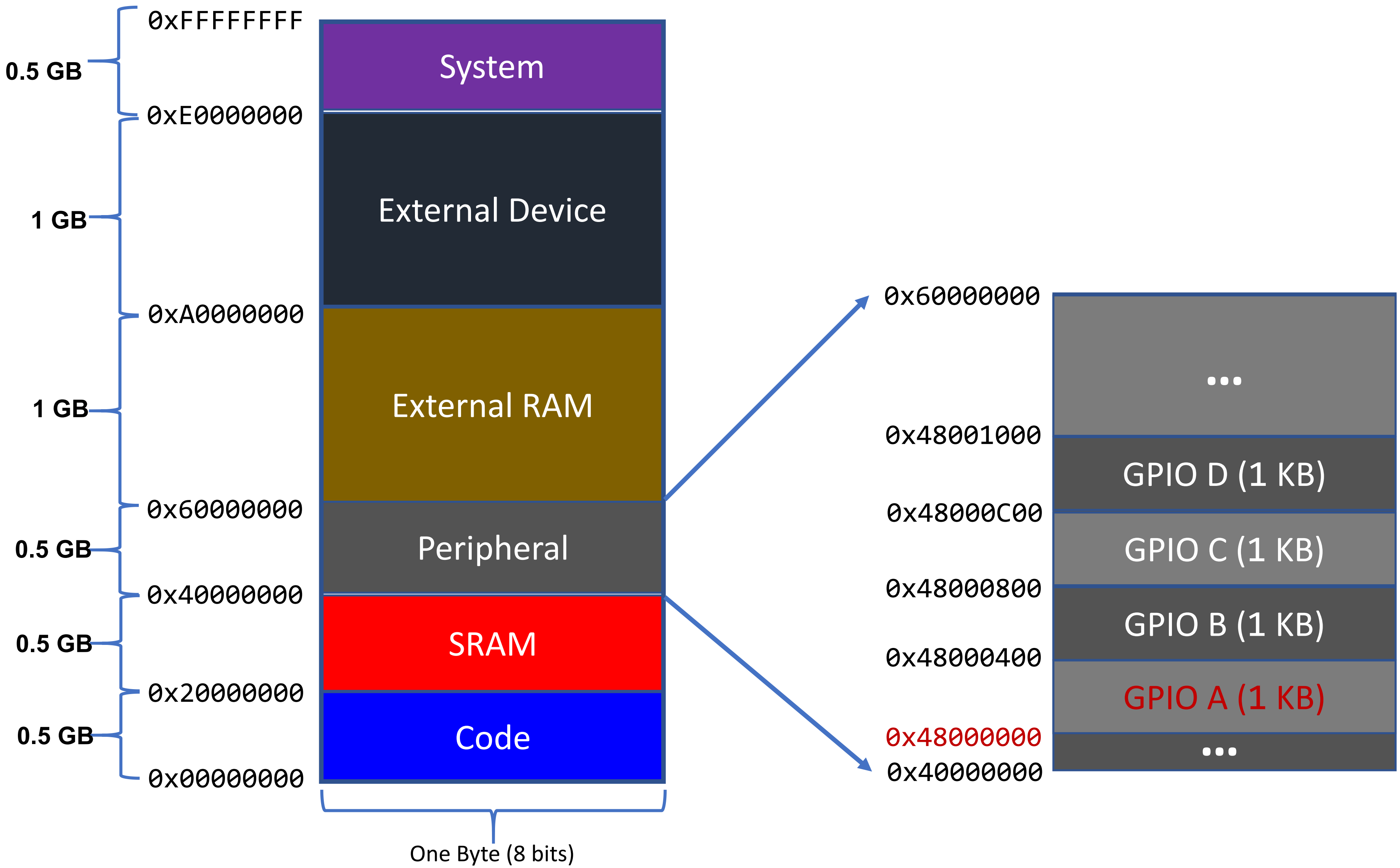


ARM Cortex-M microprocessors use memory-mapped I/O.

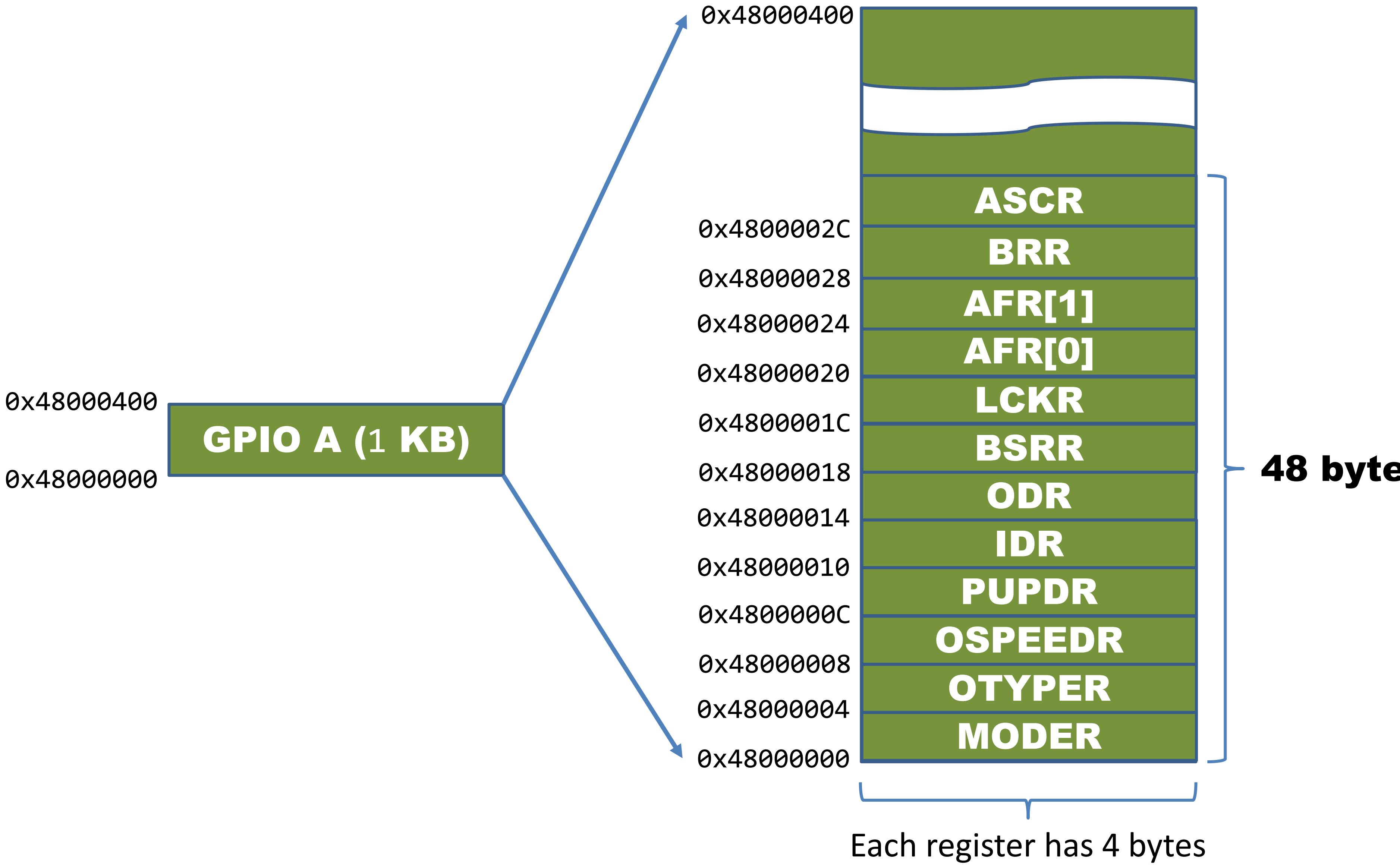
Memory Map of Cortex-M4



Memory Map of STM32L4

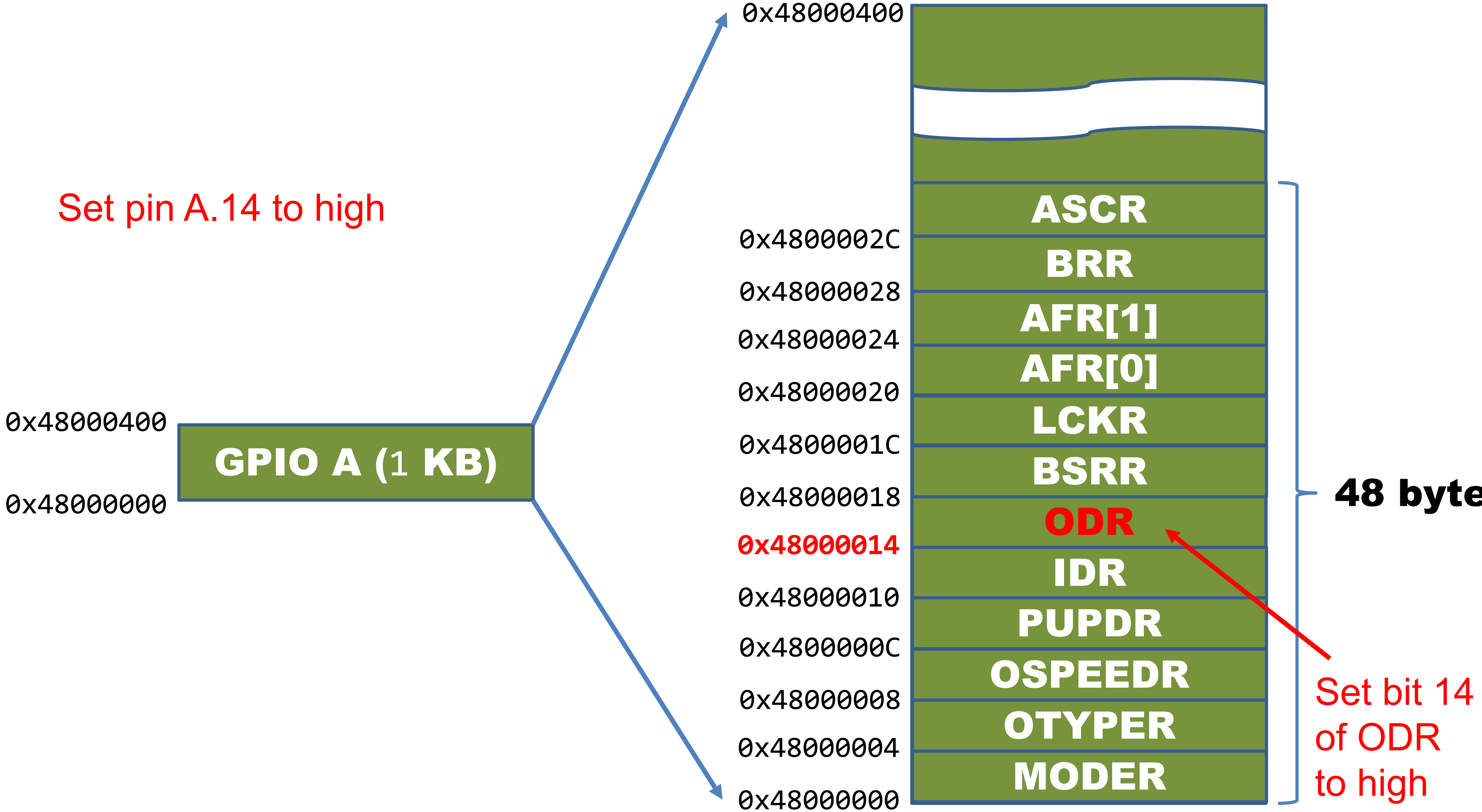


GPIO Memory Map

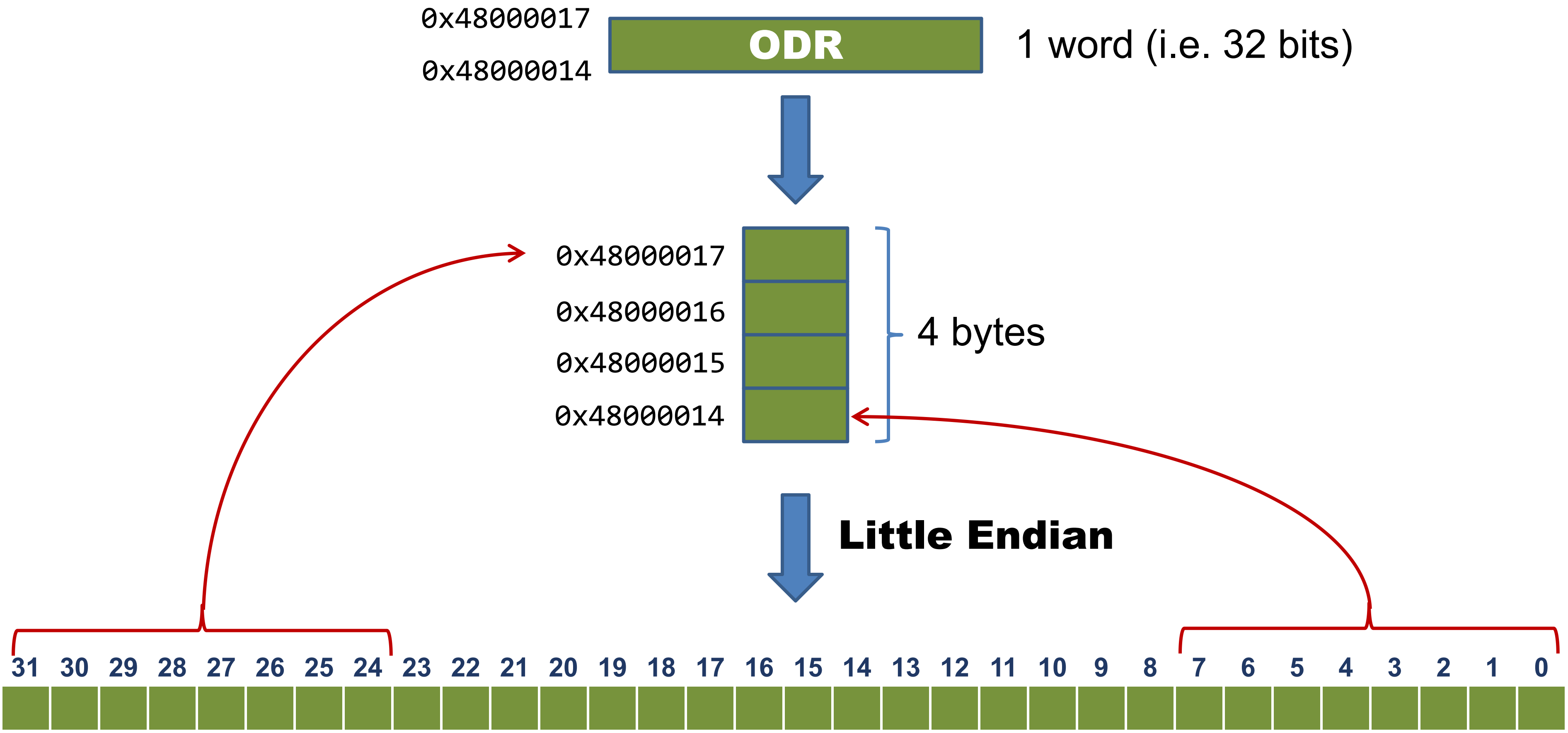


GPIO Memory Map

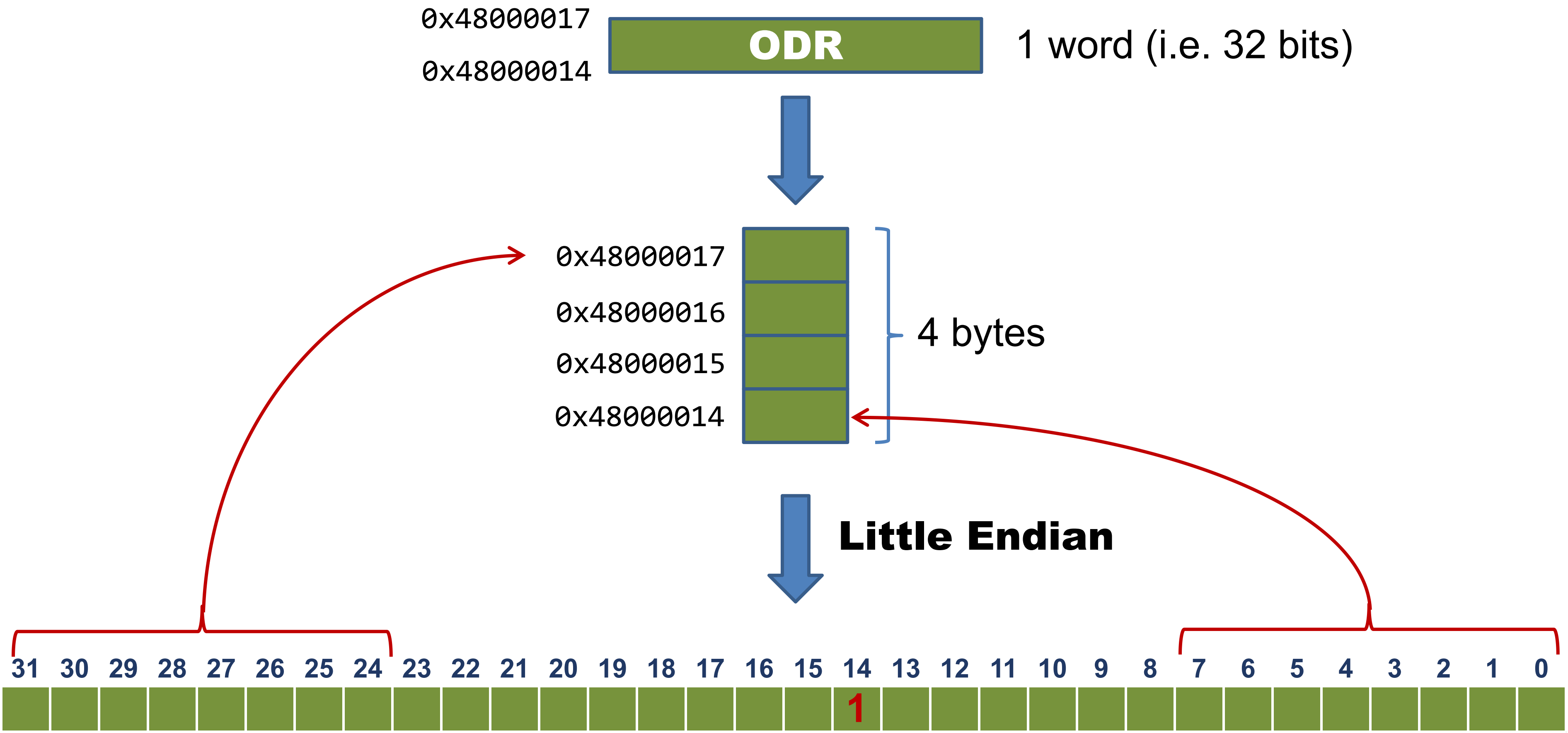
Set pin A.14 to high



Output Data Register (ODR)



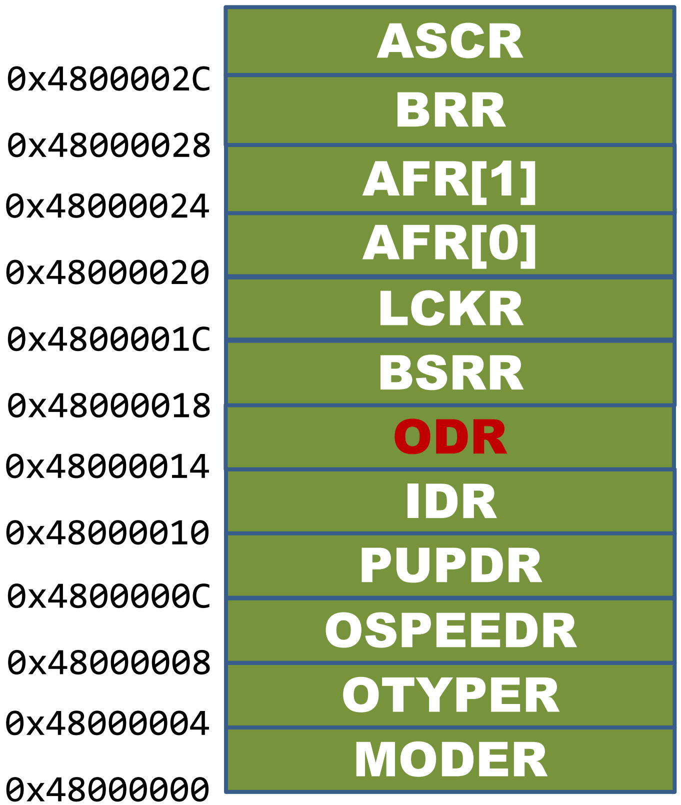
Output Data Register (ODR)




```
*((uint32_t *) 0x48000014) |= 1<<14;
```

Dereferencing a pointer **Bitwise OR**

Dereferencing a Memory Address



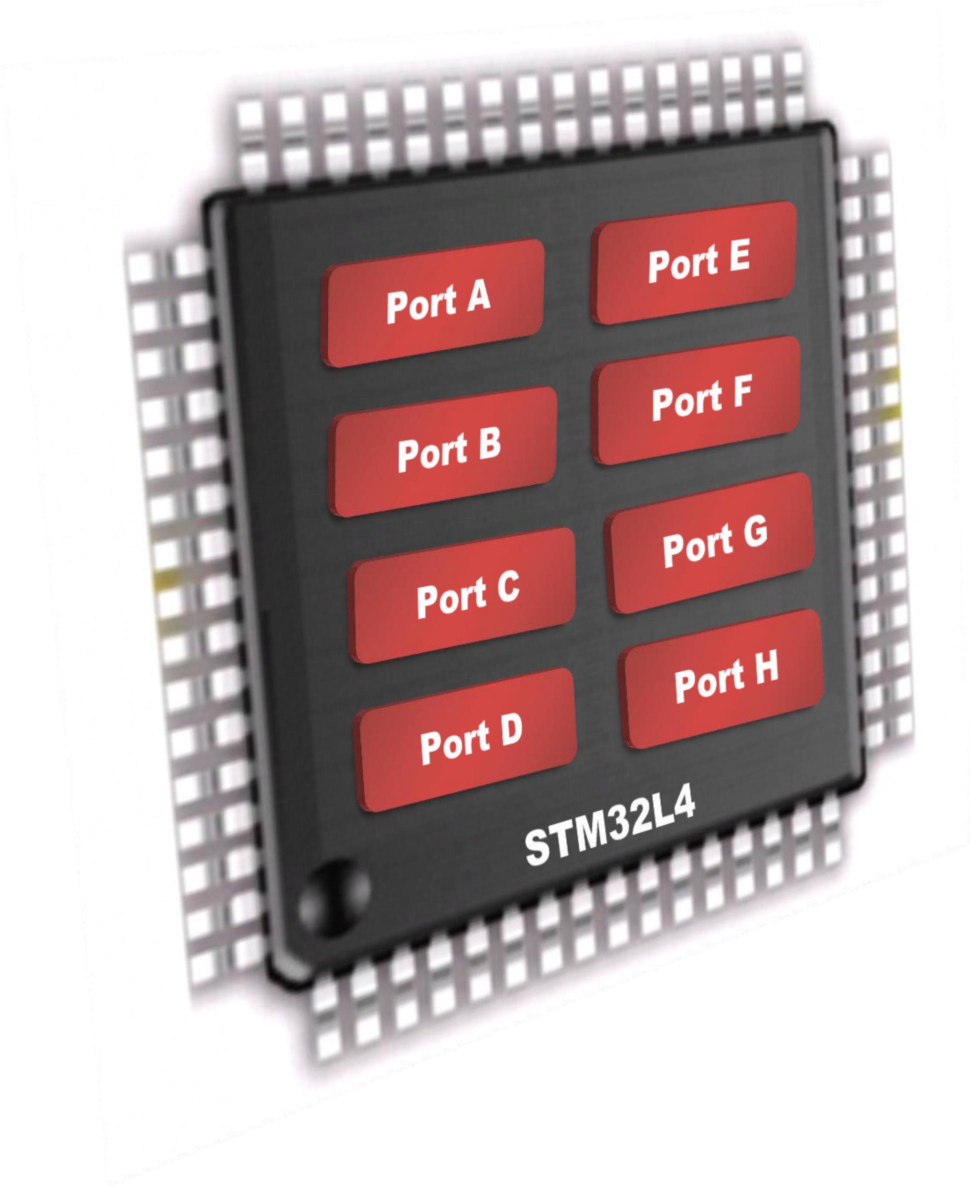

```
typedef struct {
  volatile uint32_t MODER;    // Mode register
  volatile uint32_t OTYPER;   // Output type register
  volatile uint32_t OSPEEDR;  // Output speed register
  volatile uint32_t PUPDR;    // Pull-up/pull-down register
  volatile uint32_t IDR;      // Input data register
  volatile uint32_t ODR;      // Output data register
  volatile uint32_t BSRR;     // Bit set/reset register
  volatile uint32_t LCKR;     // Configuration lock register
  volatile uint32_t AFR[2];   // Alternate function registers
  volatile uint32_t BRR;      // Bit Reset register
  volatile uint32_t ASCR;     // Analog switch control register
} GPIO_TypeDef;

// Casting memory address to a pointer
#define GPIOA ((GPIO_TypeDef *) 0x48000000)
```

```
GPIOA->ODR |= 1<<14;
```

or

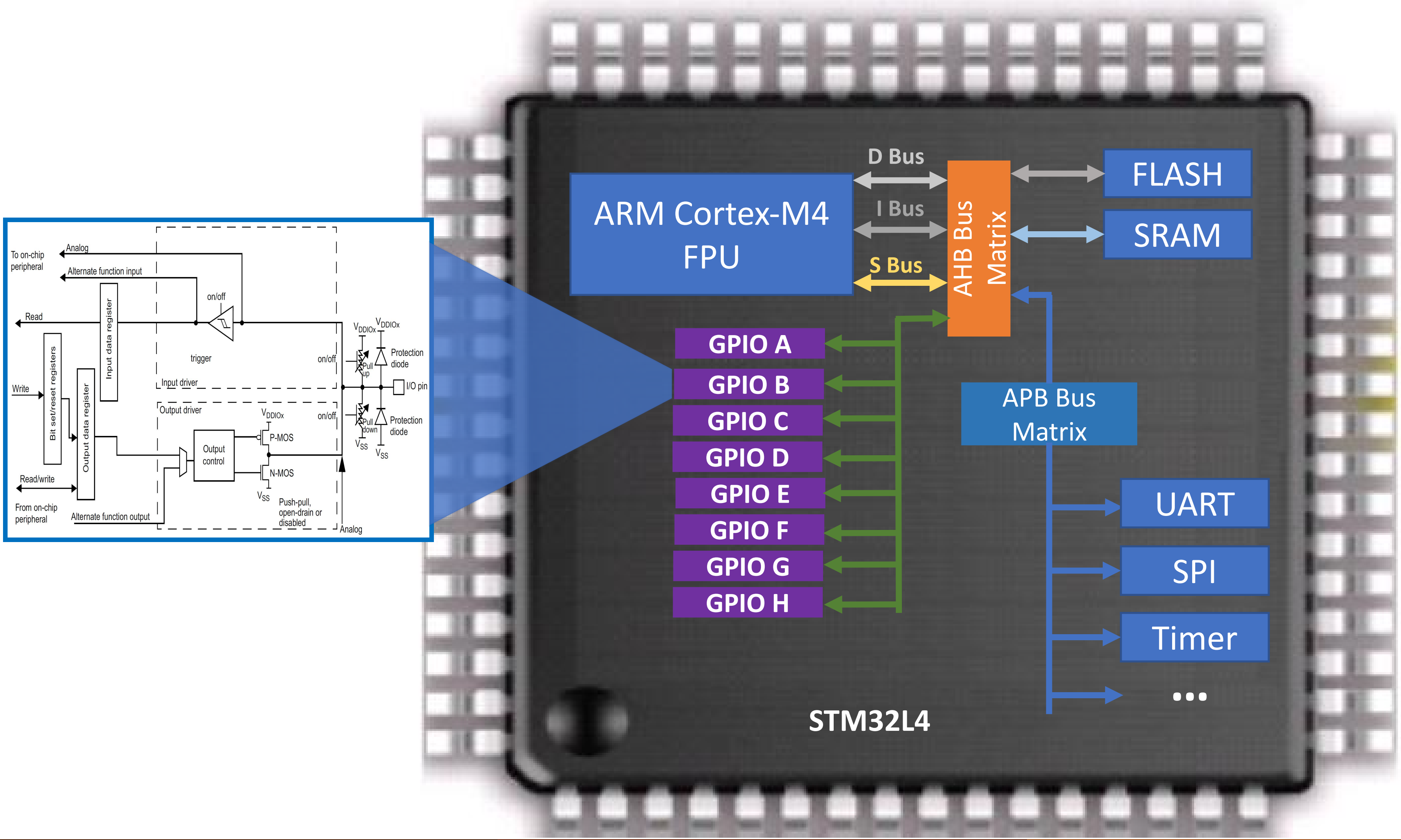
```
(*GPIOA).ODR |= 1<<14;
```

General Purpose Input/Output (GPIO)



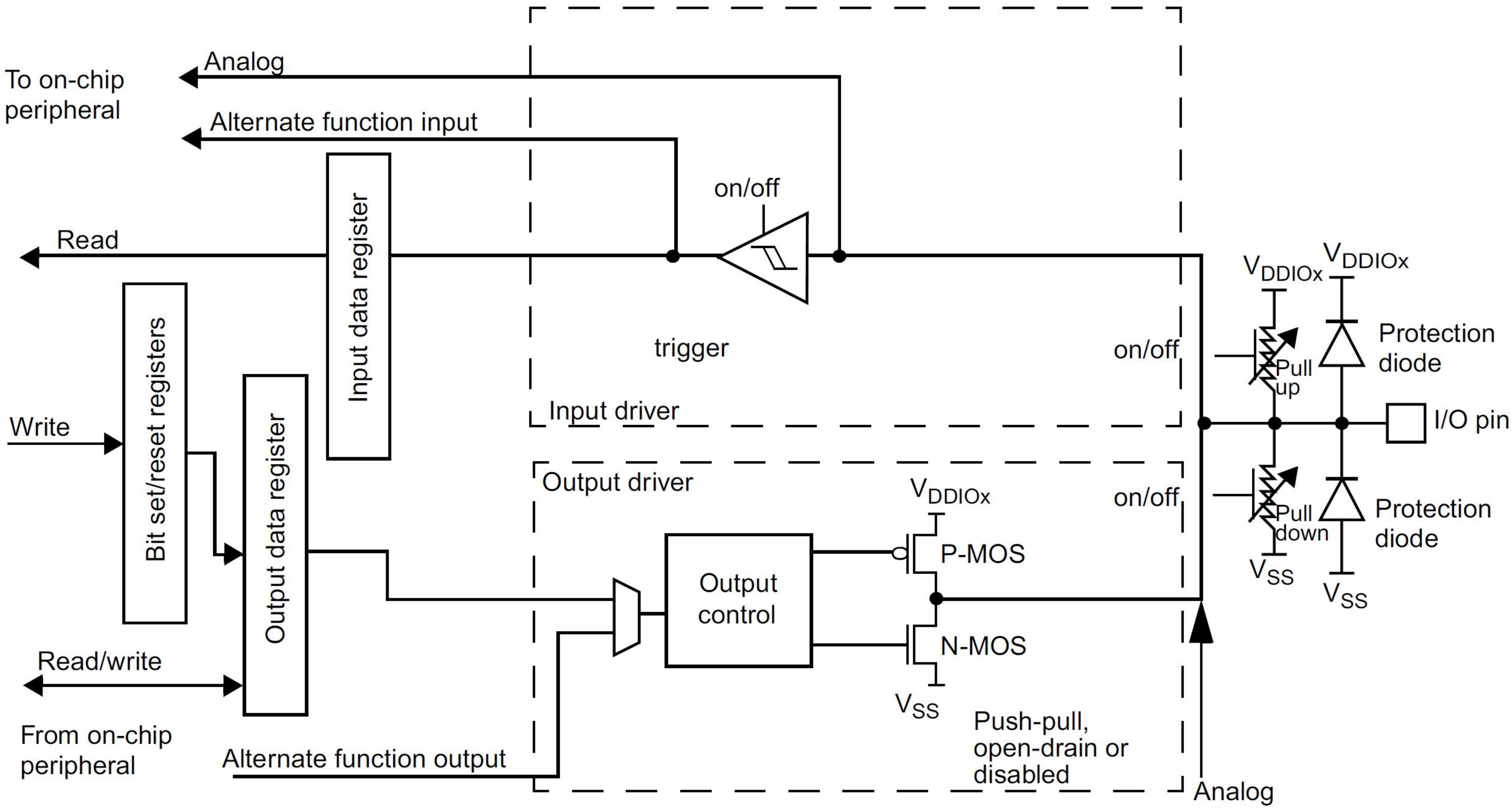
- 8 GPIO Ports:
  A, B, C, D, E, F, G, H
- Up to 16 pins in each port

General Purpose Input/Output (GPIO)



Basic Structure of an I/O Port Bit

- **Input** and **Output**:



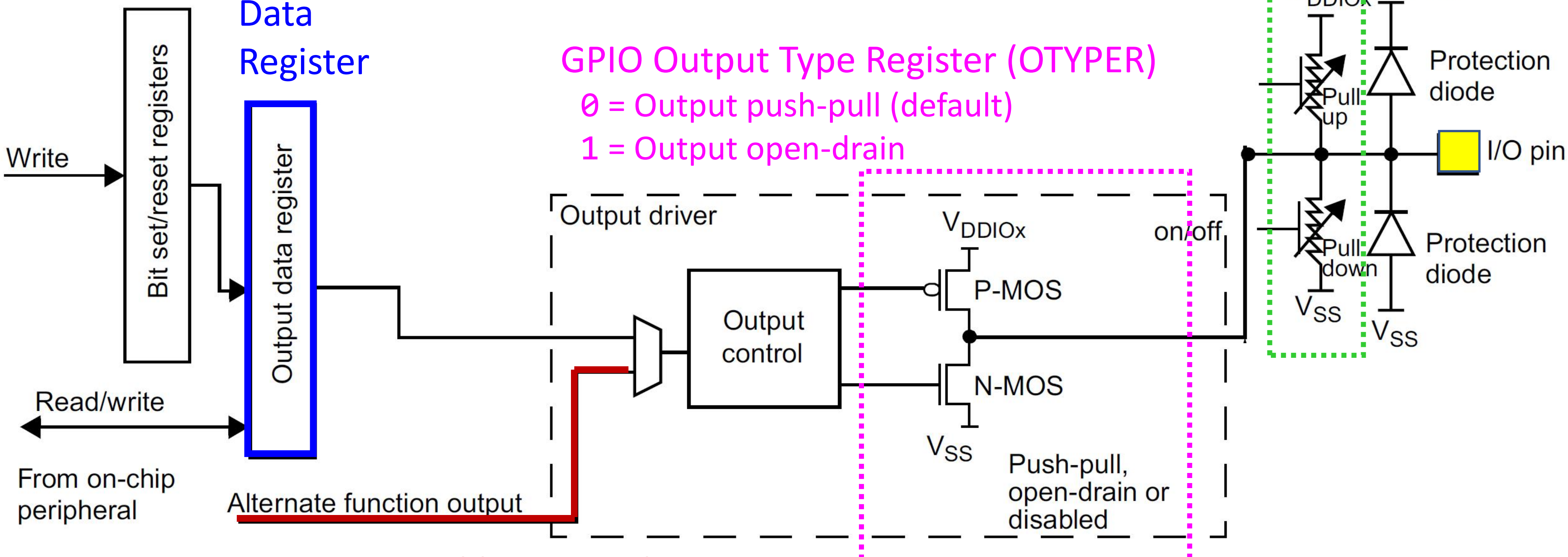
Basic Structure of an I/O Port Bit

- Only **Output**:

GPIO Pull-up/Pull-down Register (PUPDR)
00 = No pull-up, pull-down   01 = Pull-up
10 = Pull-down   11 = Reserved

Output Data Register

GPIO Output Type Register (OTYPER)
0 = Output push-pull (default)
1 = Output open-drain



GPIO MODE Register: 00 = Input, 01 = Output,
10 = AF, 11 = Analog (default)

Enable GPIO Clock

- AHB2 peripheral clock enable register (RCC_AHB2ENR)

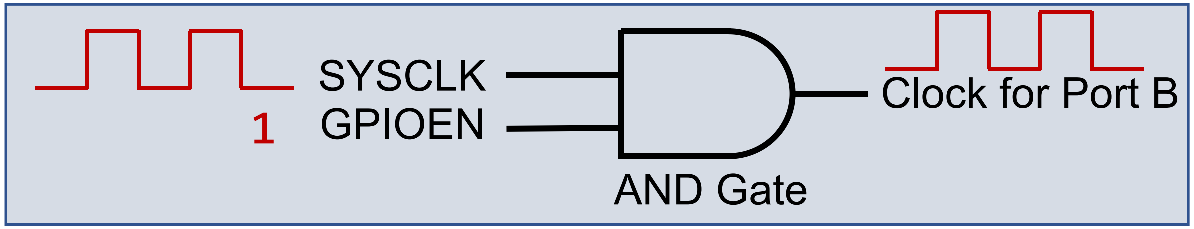
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RNG EN	Res.	AESEN
													rw		rw
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Res.	Res.	ADCEN	OTGFS EN	Res.	Res.	Res.	Res.	GPIOH EN	GPIOG EN	GPIOF EN	GPIOE EN	GPIOD EN	GPIOC EN	GPIOB EN	GPIOA EN
		rw	rw					rw	rw	rw	rw	rw	rw	rw	rw

Bit 1 **GPIOBEN:** IO port B clock enable

Set and cleared by software.

0: IO port B clock disabled

1: IO port B clock enabled




```
#define  RCC_AHB2ENR_GPIOBEN  ((uint32_t)0x00000002U)

RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;
```

GPIO Mode Register (MODER)

- 32 bits (16 pins, 2 bits per pin)

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

Bits 2y+1:2y **MODEy[1:0]:** Port x configuration bits (y = 0..15)

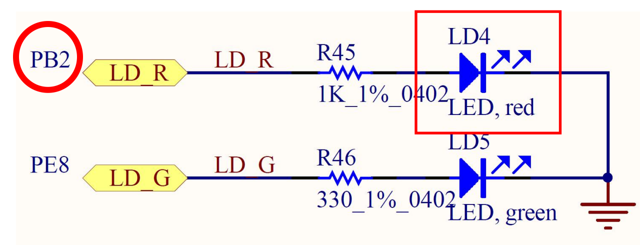
These bits are written by software to configure the I/O mode.

00: Input mode

01: General purpose output mode

10: Alternate function mode

11: Analog mode (reset state)




```
GPIOB->MODER &= ~(3<<4);   // Clear bits 4 and 5 for Pin 2
GPIOB->MODER |=   1<<4;    // Set bit 4, set Pin 2 as output
```

GPIO Output Type Register (OTYPER)

- 16 bits reserved, 16 data bits, 1 bit for each pin

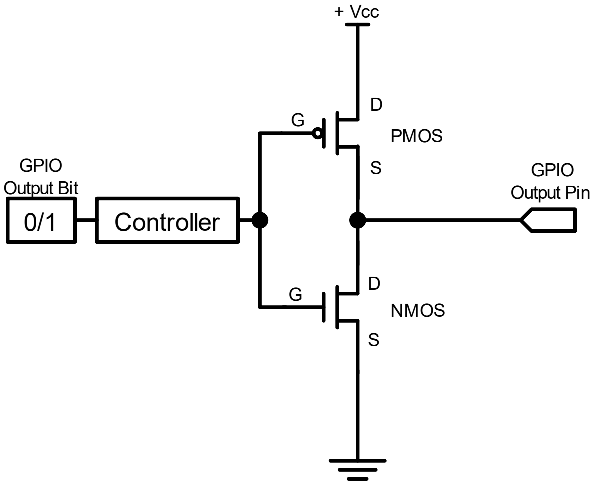
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw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

Bits 15:0 **OTy:** Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O output type.

0: Output push-pull (reset state)

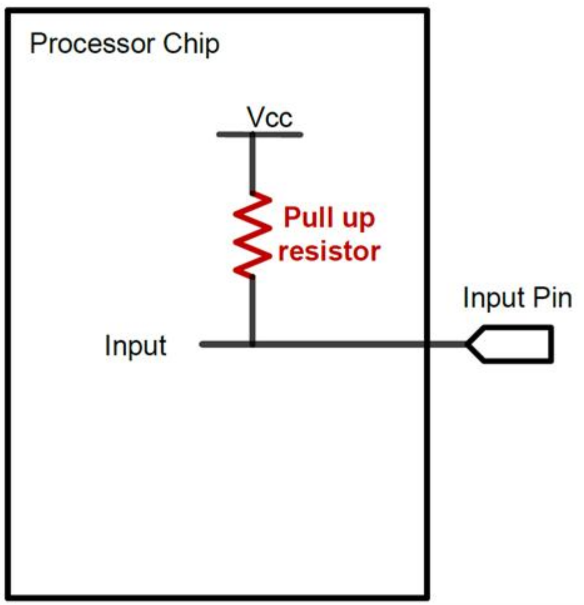
1: Output open-drain




```
GPIOB->OTYPER &= ~(1<<2);   // Clear bit 2
```

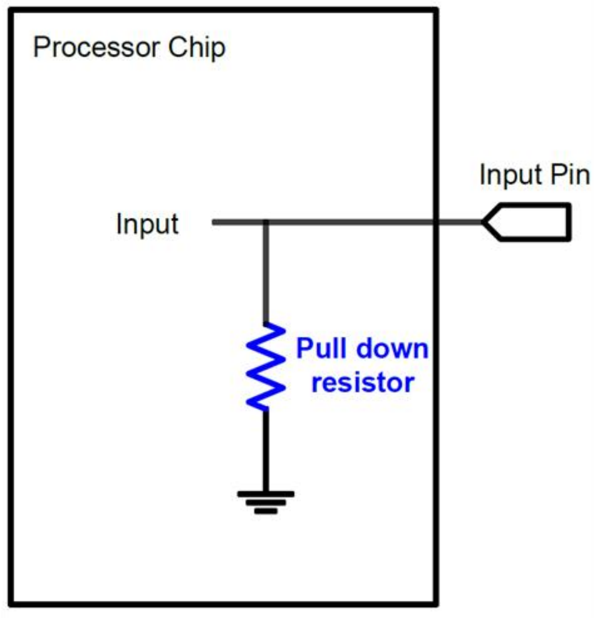
GPIO Input

- Pull Up and Pull Down:
  - A digital input can have three states: High, Low, and High-Impedance (also called floating, tri-stated, HiZ)



Pull-Up

If external input is HiZ, the input is read as a valid HIGH.

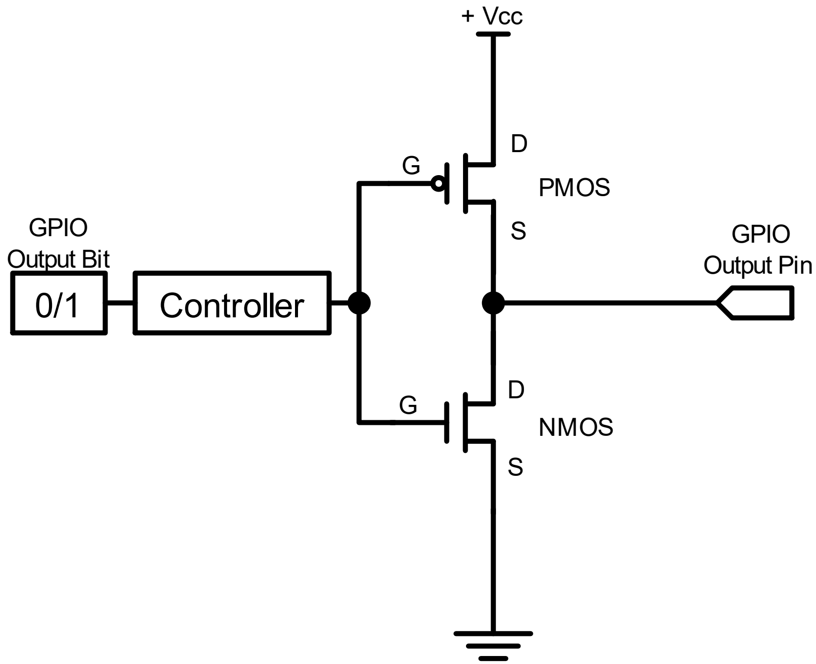


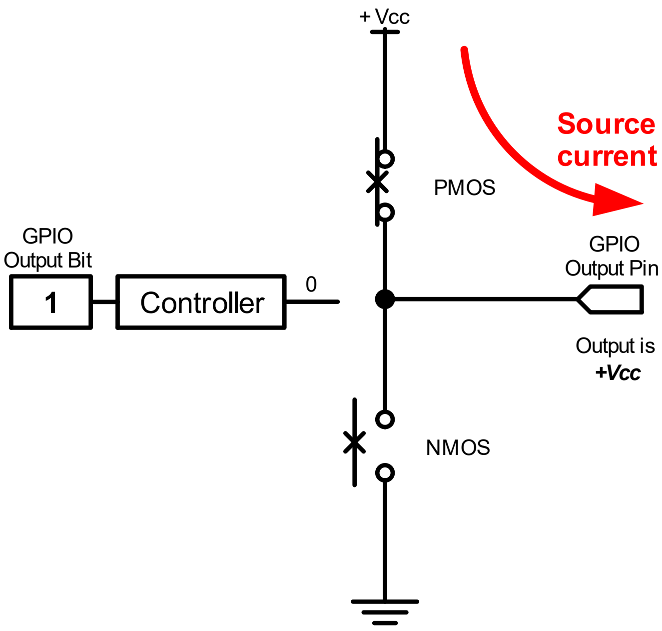
Pull-Down

If external input is HiZ, the input is read as a valid LOW.

GPIO Output

- Push-pull:

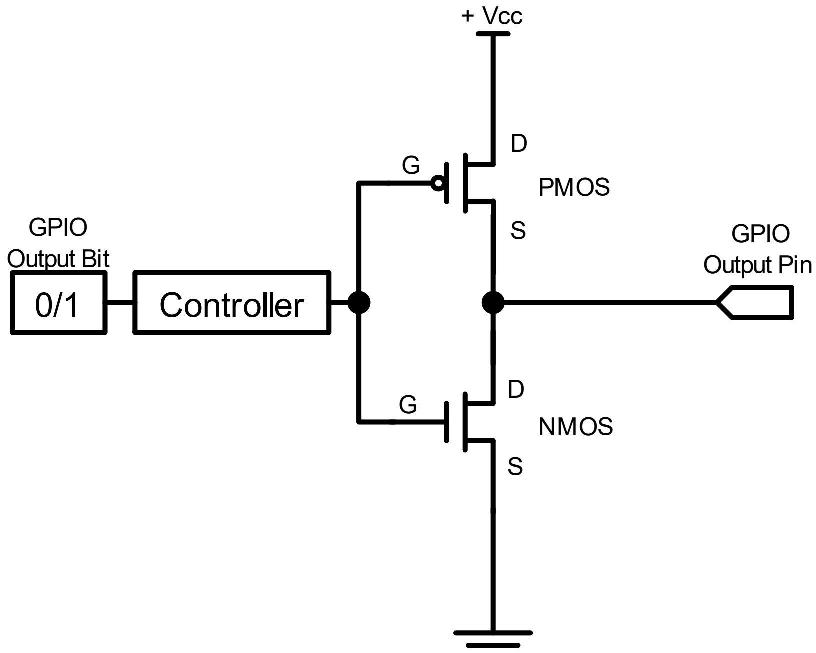


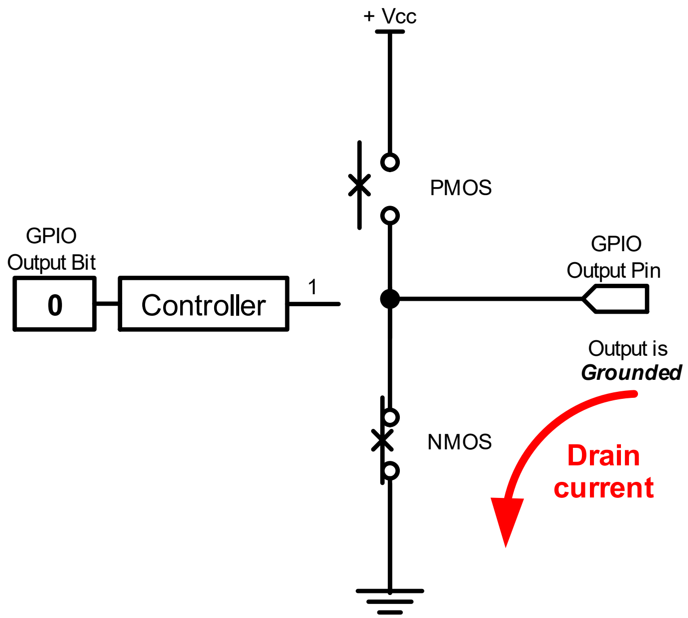


GPIO Output = 1

Source current to external circuit

- Push-pull:

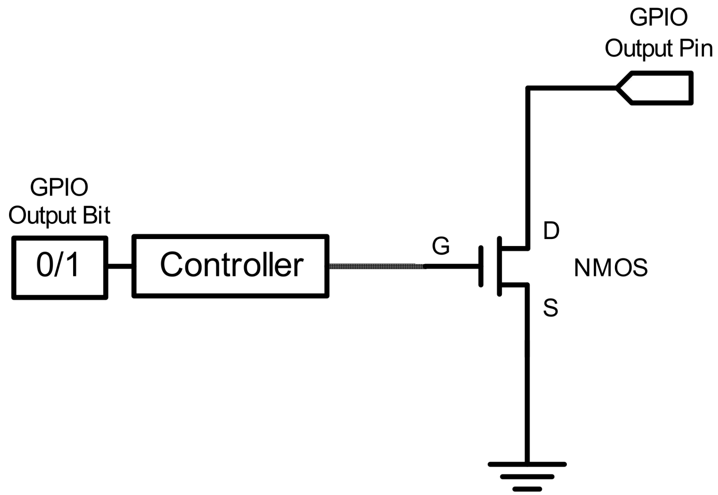


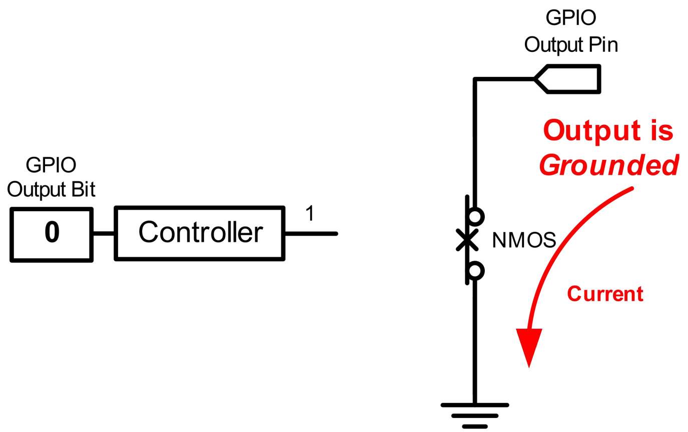


GPIO Output = 0

Drain current from external circuit

- Open-drain:

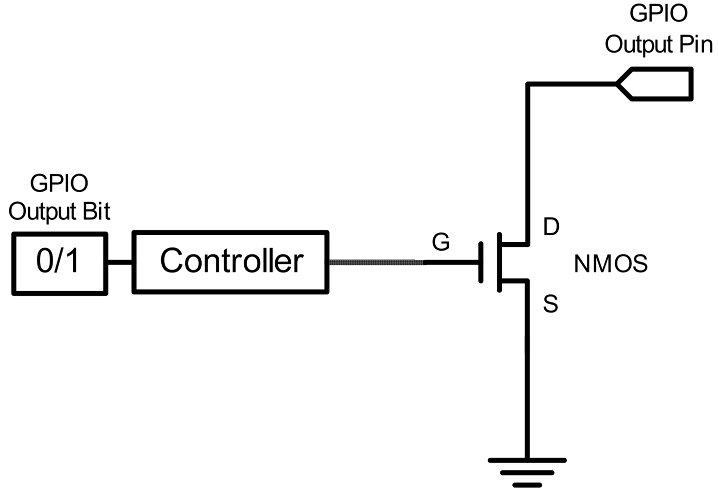


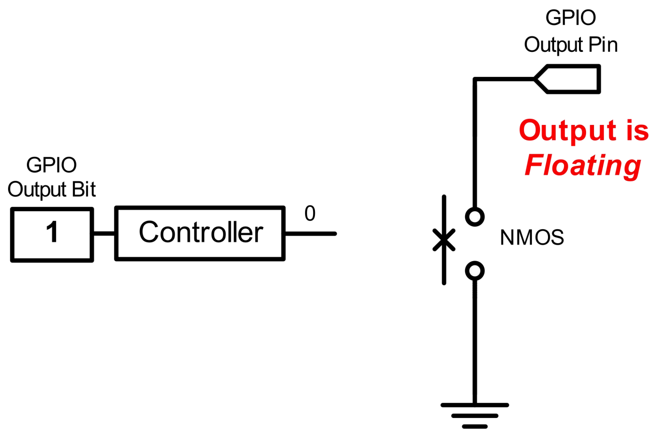


GPIO Output = 0
Drain current from external circuit

GPIO Output

- Open-drain:

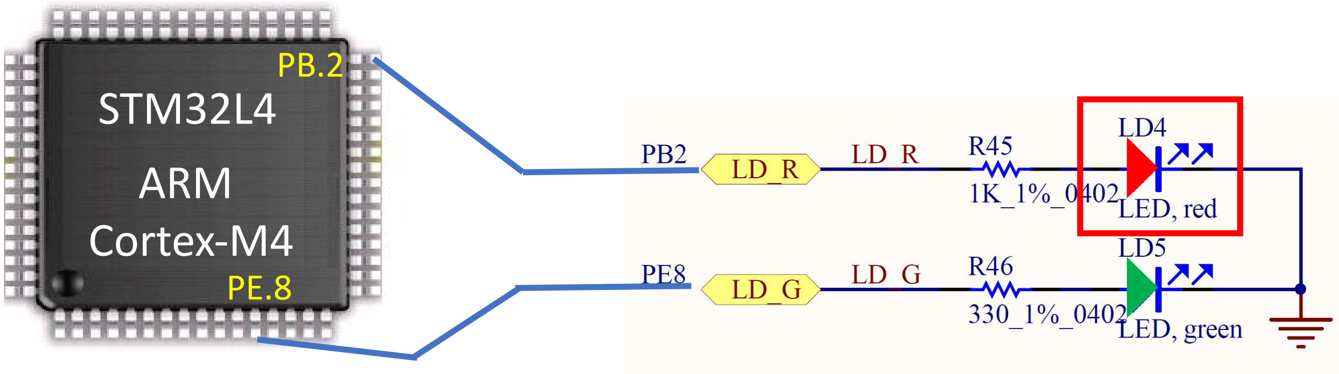




Output = 1

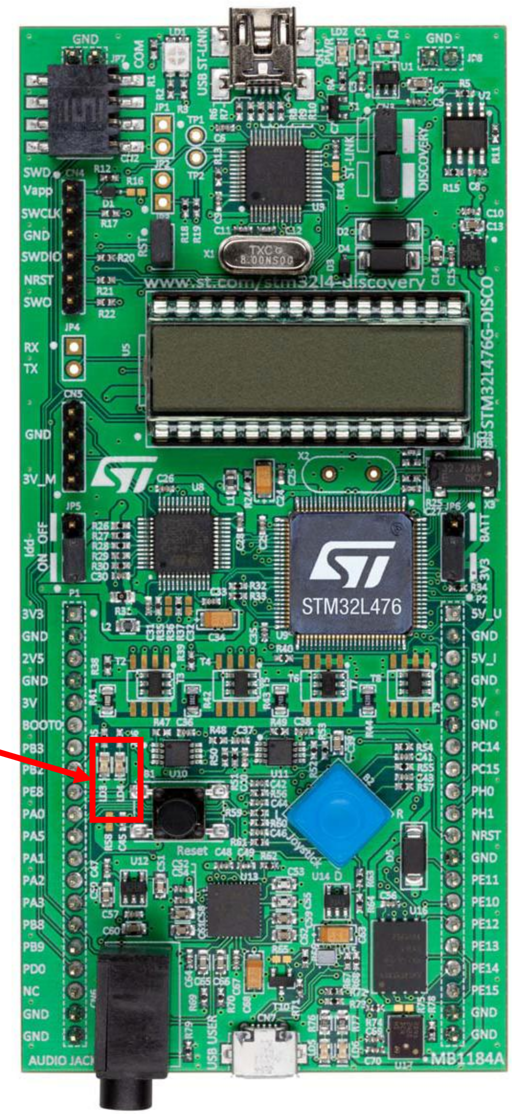
GPIO Pin has high-impedance to external circuit

GPIO Output: Push-pull vs Open-Drain



PB.2	Red LED
High	On
Low	Off

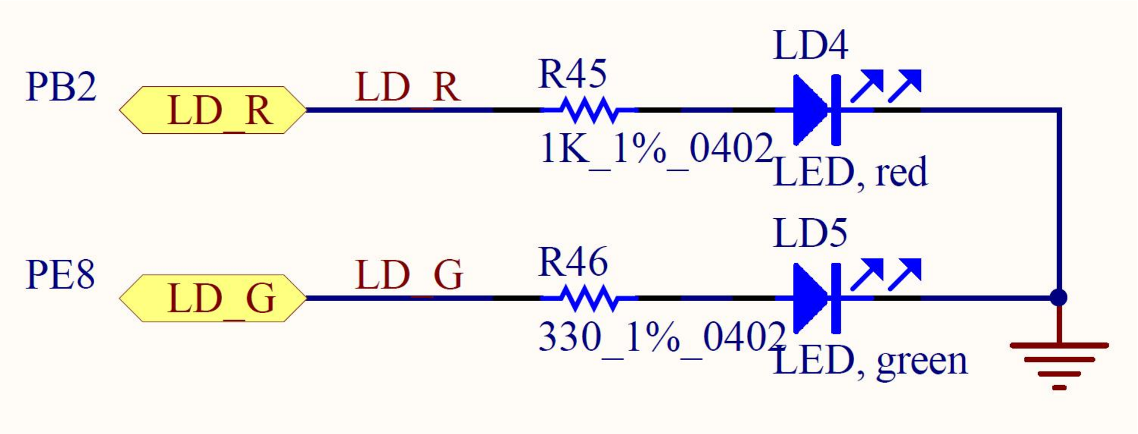
Red & Green LEDs



GPIO Output: Push-pull vs Open-Drain

	Voltage Level	
Output Bit	Push-Pull	Open-Drain
1	High	HiZ
0	Low	Low

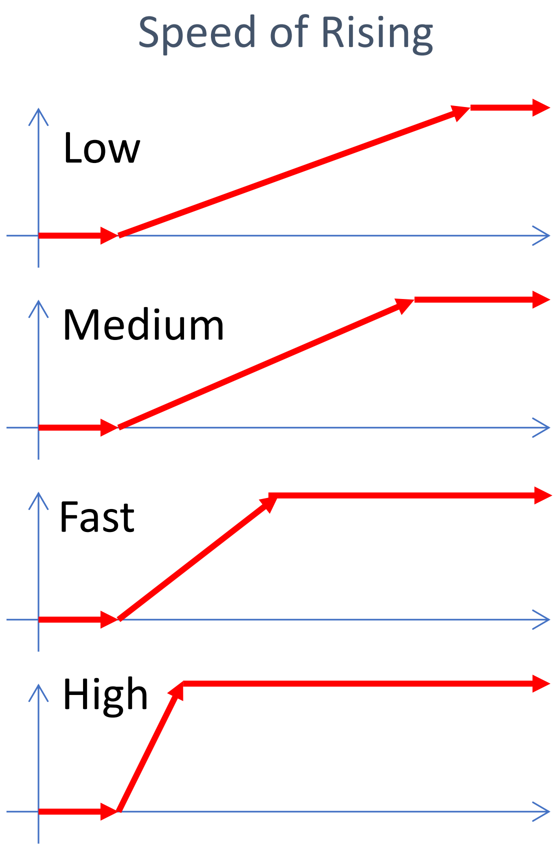
Note: HiZ → High-impedance



Use push-pull output, instead of open-drain output!

GPIO Output Speed

- Output Speed:
  - Speed of rising and falling
  - Four speeds: Low, Medium, Fast, High
- Tradeoff
  - Higher GPIO speed increases EMI noise and power consumption
  - Configure based on peripheral speed
    - Low speed for toggling LEDs
    - High speed for SPI

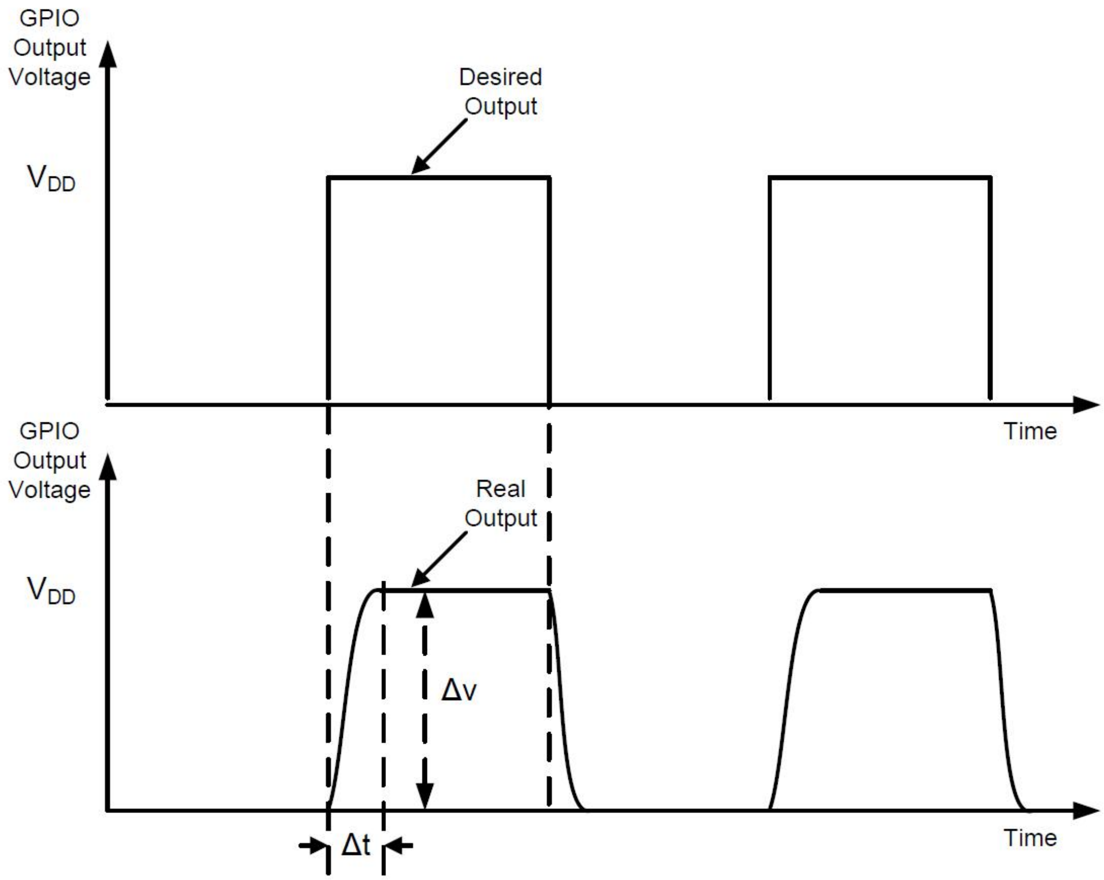


Slew Rate

Slew Rate:
Maximum rate of change of the output voltage

$$Slew\ Rate = max\left(\frac{\Delta V}{\Delta t}\right)$$

A high slew rate allows the output to be toggled at a fast speed.



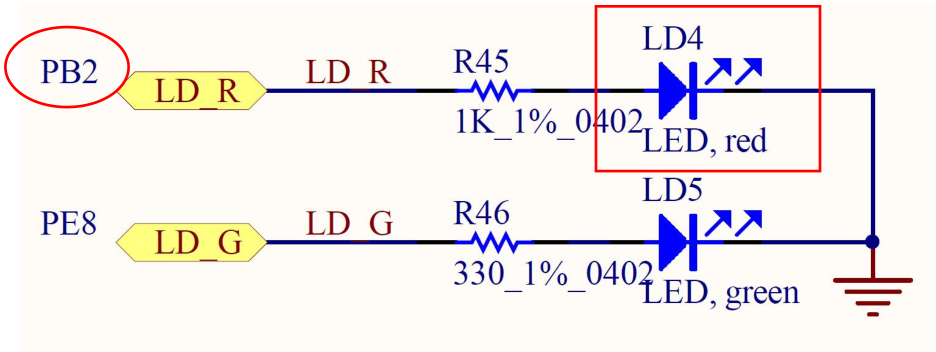
GPIO Output Data Register (ODR)

- 16 bits reserved, 16 data bits, 1 bit for each pin

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

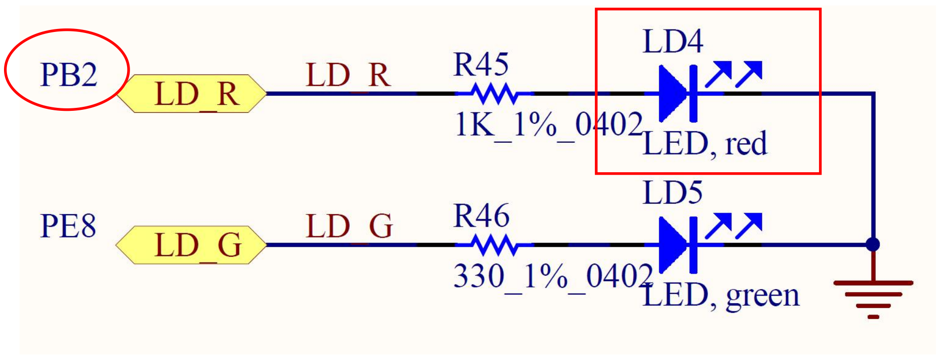
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OD15	OD14	OD13	OD12	OD11	OD10	OD9	OD8	OD7	OD6	OD5	OD4	OD3	OD2	OD1	OD0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Pin 2




```
GPIOB->ODR |= 1 << 2;     // Set bit 2
```

Light up the Red LED (PB.2)




```
RCC->AHB2ENR |= 0x02;   // Enable clock of Port B

GPIOB->MODER &= ~(3<<4); // Clear mode bits
GPIOB->MODER |=   1<<4;  // Set mode to output

GPIOB->OTYPER &= ~(1<<2); // Select push-pull output

GPIOB->ODR   |= 1 << 2;  // Output 1 to turn on red LED
```

- All GPIO registers can be found on D2L under the section “Register Maps and Processor Manual”.

RM0351 General-purpose I/Os (GPIO)

9.4.13 GPIO register map

The following table gives the GPIO register map and reset values.

Table 33. GPIO register map and reset values

Offset	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
0x00	GPIOA_MODER	MODE15[1:0]		MODE14[1:0]		MODE13[1:0]		MODE12[1:0]		MODE11[1:0]		MODE10[1:0]		MODE9[1:0]		MODE8[1:0]		MODE7[1:0]		MODE6[1:0]		MODE5[1:0]		MODE4[1:0]		MODE3[1:0]		MODE2[1:0]		MODE1[1:0]		MODE0[1:0]	
	Reset value	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0x00	GPIOB_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]	
	Reset value	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1
0x00	GPIOx_MODER (where x = C..H)	MODE15[1:0]		MODE14[1:0]		MODE13[1:0]		MODE12[1:0]		MODE11[1:0]		MODE10[1:0]		MODE9[1:0]		MODE8[1:0]		MODE7[1:0]		MODE6[1:0]		MODE5[1:0]		MODE4[1:0]		MODE3[1:0]		MODE2[1:0]		MODE1[1:0]		MODE0[1:0]	
	Reset value	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0x04	GPIOx_OTYPER (where x = A..H)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

In-lab Assignment 1

- **This assignment is not a test! You can ask any question to TAs!**
  - You will receive 25 points if you answer all questions! Even if the answer is not correct!
  - If you don't try to answer all questions, you will receive partial credits!
  - It must be finished on this class!

For Next Class

- **Lab 0 – Homework 2 is due next class!** You can download it on D2L! There will be no Dropbox submission for this assignment!

- We are going to have a Hands-On lab next class!

Lab 0: Hand-On Lab

Graduate Teaching Assistants:

Francisco E. Fernandes Jr.

feferna@okstate.edu

Khuong Vinh Nguyen

Khuong.V.Nguyen@okstate.edu

School of Electrical and Computer Engineering

Oklahoma State University

Spring 2019

Overview

- **Today:**
  - Create a project from scratch in System Workbench for STM32.
  - Show and explain a simple C code to turn ON both RED and GREEN LEDs.
  - Show and explain a simple Assembly code to turn ON both RED and GREEN LEDs.
  - **In-lab assignment:** all students must write, compile and test the given code by themselves.
- **Next class:**
  - **Pre-lab 1 (10 points) is due next class!**
    - **For Mondays students:** February 11, 2019.
    - **For Wednesday students:** February 13, 2019.