Lab 2: Interfacing Joystick and LEDs

Instructor:
Dr. Carl Latino
carl.latino@okstate.edu

Graduate Teaching Assistant:
Francisco E. Fernandes Jr.
feferna@okstate.edu

School of Electrical and Computer Engineering
Oklahoma State University

Fall 2019

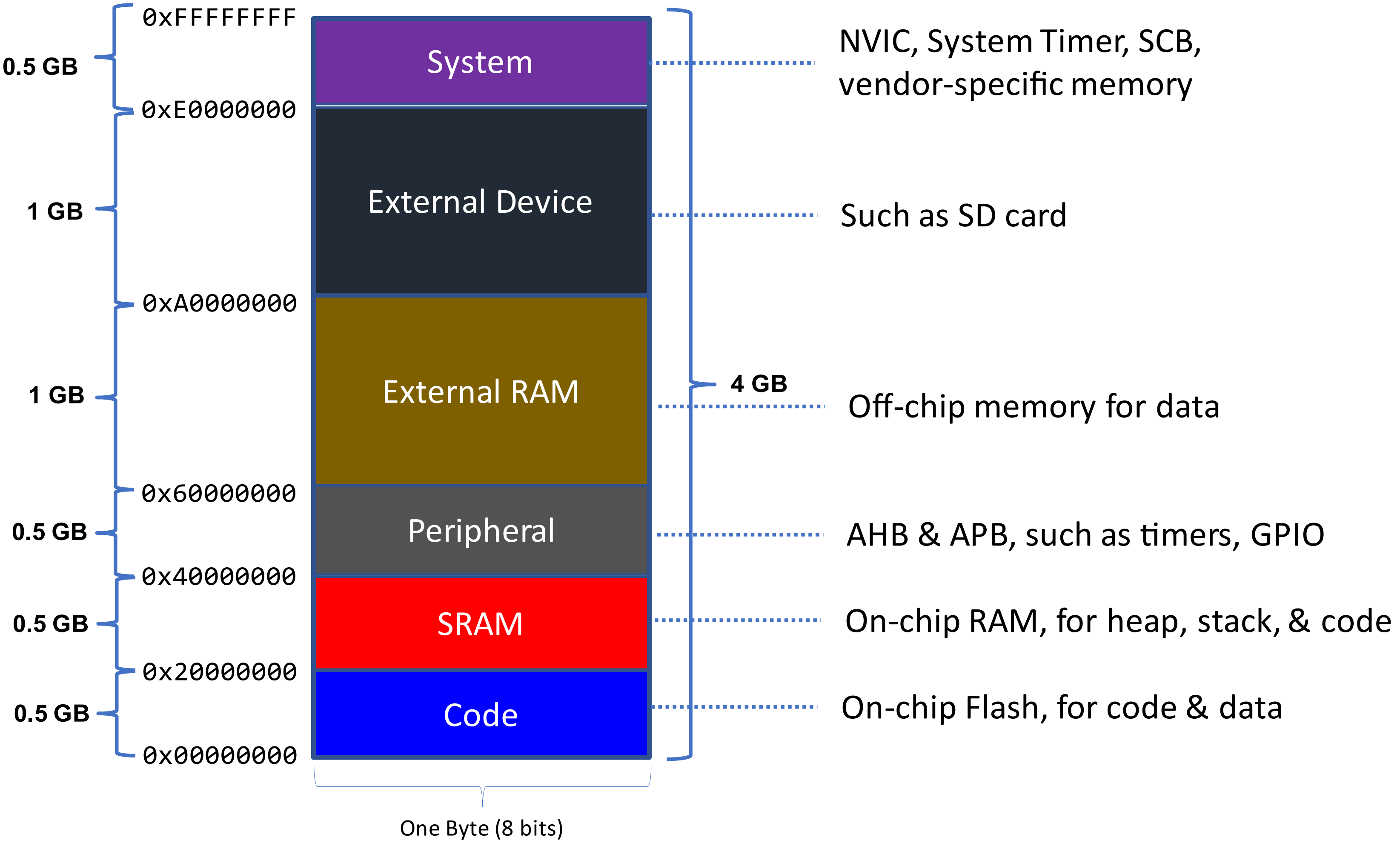
Overview

- **Introduction to General Purpose Input and Output (GPIO):**
  - **GPIO operation modes.**
  - **GPIO registers.**
- **Lab Assignment:**
  - **Write an Assembly program that uses the onboard joystick to control both the red and green LEDs as follows:**
    - **If the UP button is pushed, TURN ON both LEDs.**
    - **If the DOWN button is pushed, TURN OFF both LEDs.**

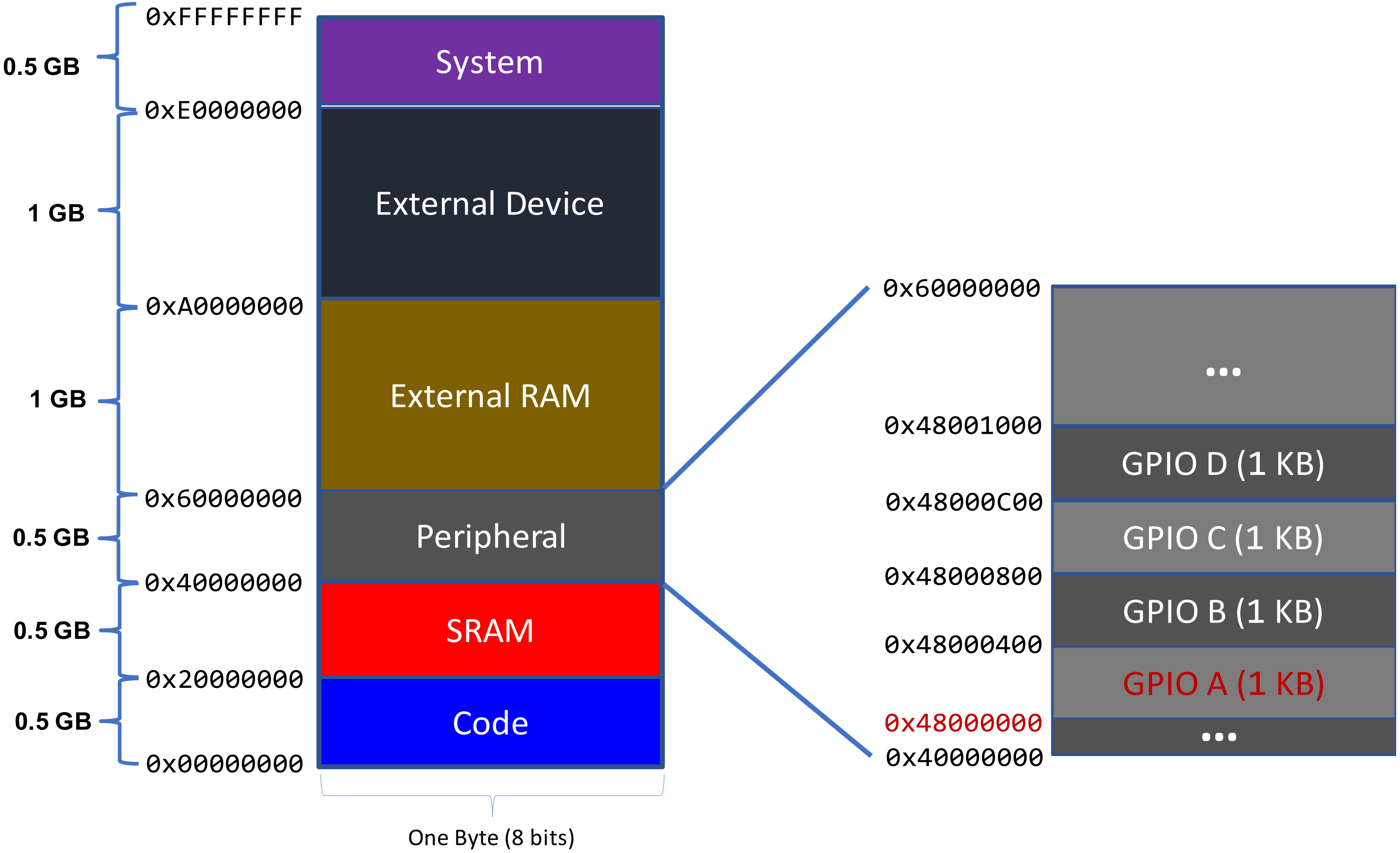
Schedule and Grading

- **Lab 2** will take a total of **two weeks**:
  - **September 16, 2019:**
    - Complete and show to the T.A. the code to initialize the GPIOs clocks and pins.
    - Use your pre-lab quiz to help you with the code!
  - **September 23, 2019:**
    - Complete and demo to the T.A. your final **WORKING** lab.
- **Grading for this lab:**
  - **Pre-lab quiz:** 2 points.
  - **Assembly code:** 8 points.
    - **GPIOs initialization:** 3 points.
    - **Working lab in the final week:** 5 points.
  - **Total: 10 points.**
- **Grading penalization:**
  - Students who disrupt the lecture by talking and not paying attention will loss 2 points in their lab 2's grade!
  - Students who do not follow the lab safety procedures (e.g. coming to lab with shorts and flip flops) will loss 1 points in their lab 2's grade!

Introduction to GPIOs – Memory Map of Cortex M4



Introduction to GPIOs – Memory Map of STM32L4



0xFFFFFFFF

0.5 GB — System

0xE0000000

1 GB — External Device

0xA0000000

1 GB — External RAM

0x60000000

0.5 GB — Peripheral

0x40000000

0.5 GB — SRAM

0x20000000

0.5 GB — Code

0x00000000

One Byte (8 bits)

0x60000000

...

0x48001000

GPIO D (1 KB)

0x48000C00

GPIO C (1 KB)

0x48000800

GPIO B (1 KB)

0x48000400

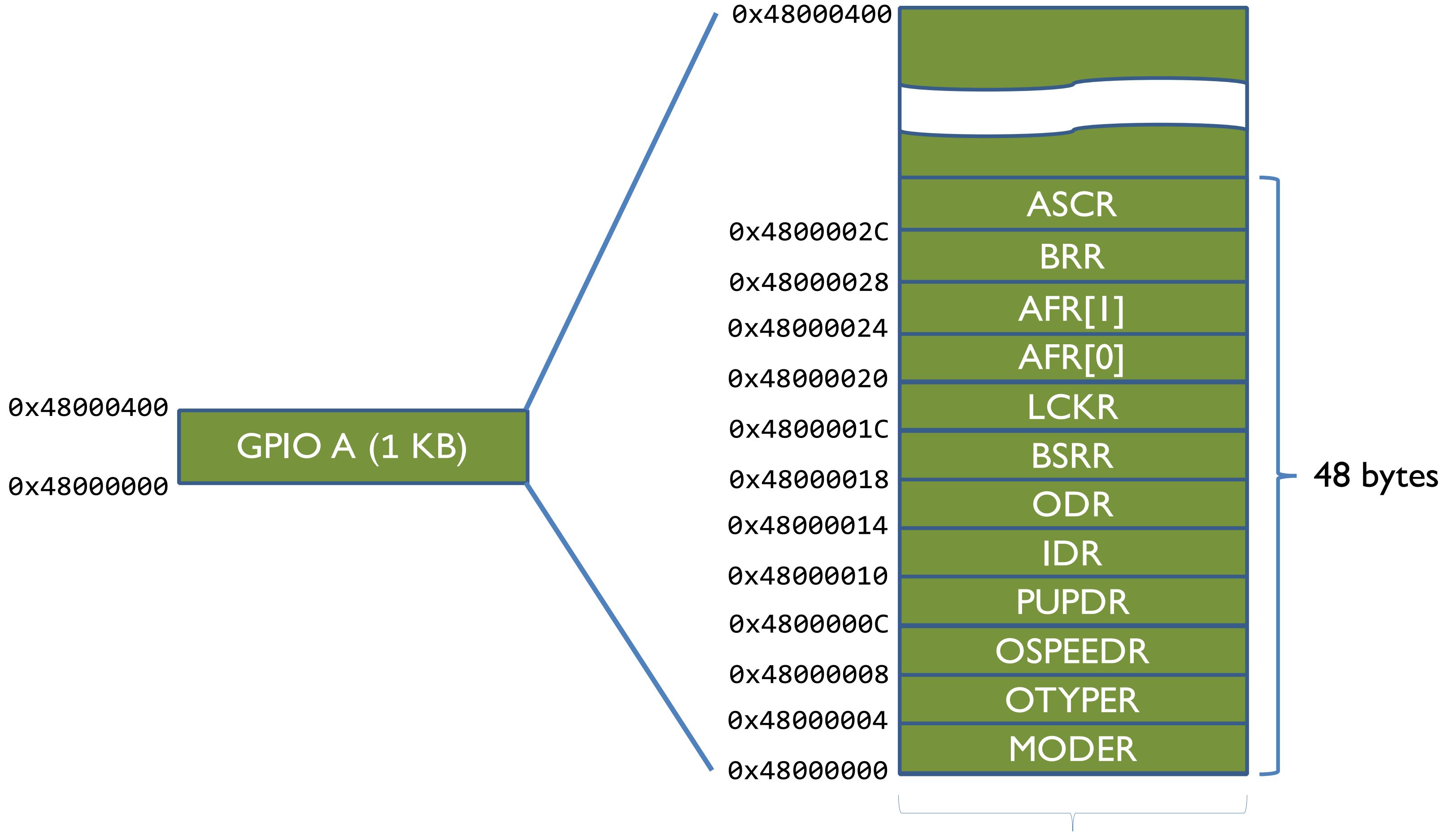
GPIO A (1 KB)

0x48000000

...

0x40000000

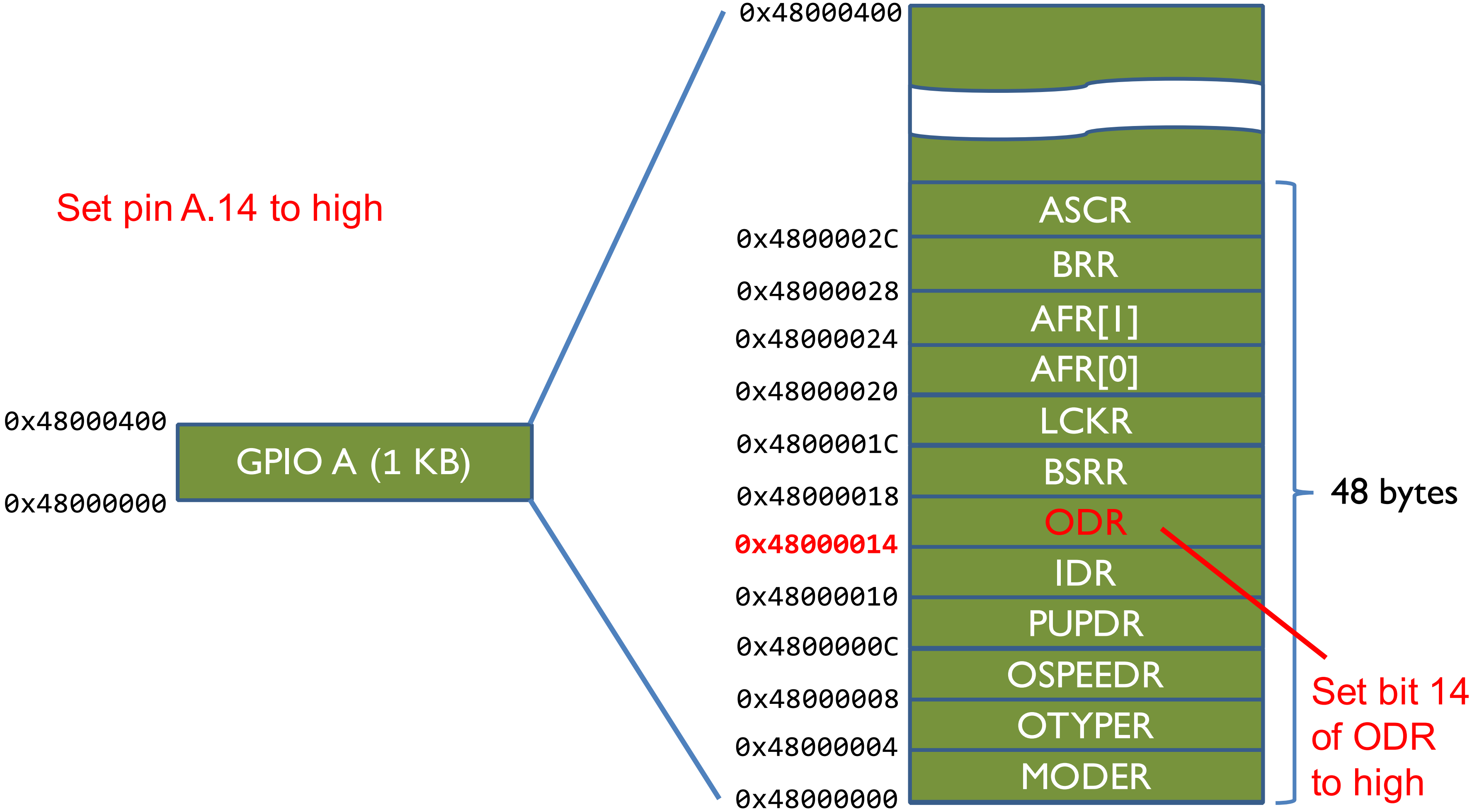
Introduction to GPIOs – GPIO Memory Map



Each register has 4 bytes

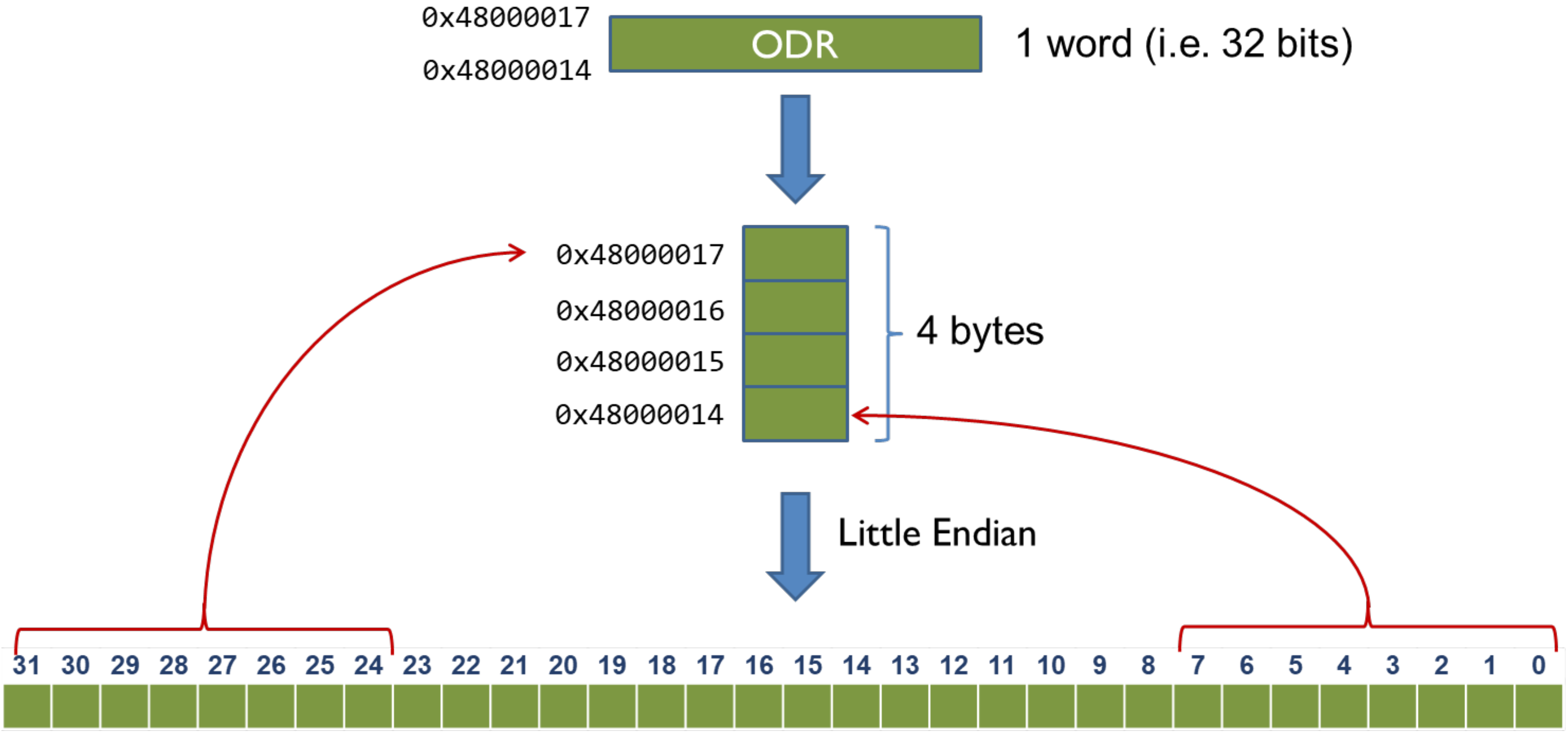
Introduction to GPIOs – GPIO Memory Map

Set pin A.14 to high

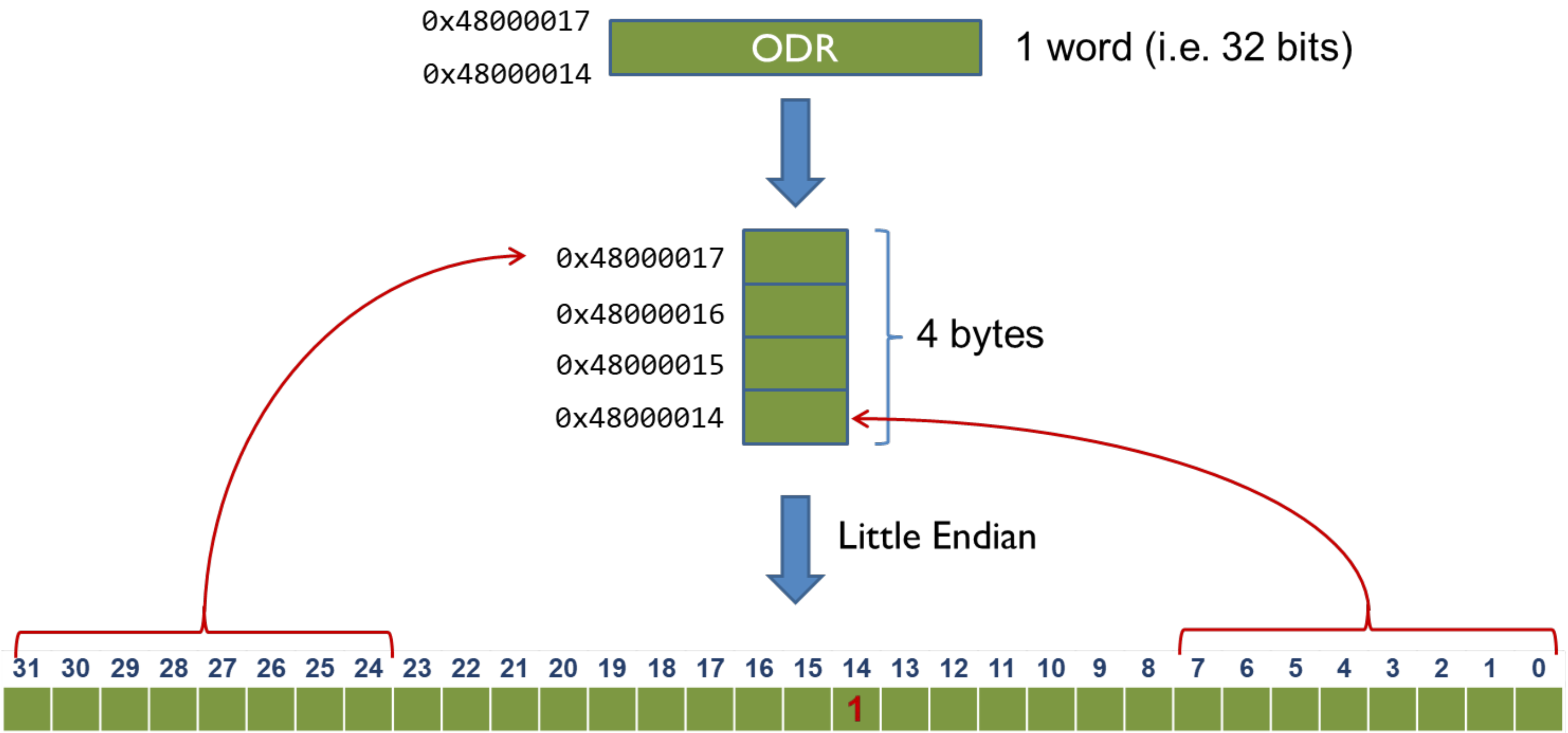


Set bit 14 of ODR to high

Introduction to GPIOs – Output Data Register



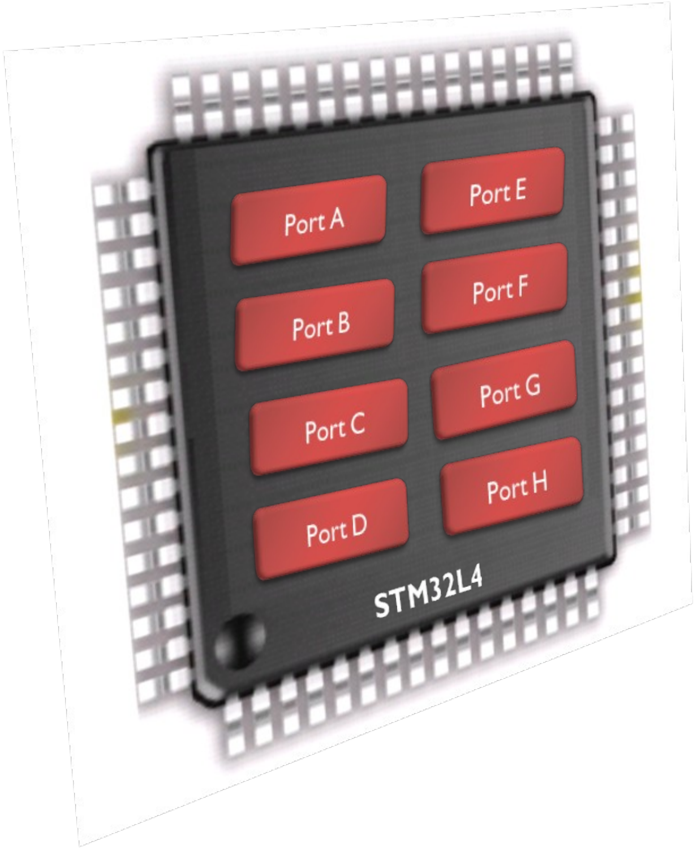
Introduction to GPIOs – Output Data Register



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
																	1														

```
LDR r0, =0x48000014       // r0 = 0x48000014
LDR r1, [r0]              // r1 = the memory contents located at 0x48000014
LDR r2, =0x4000           // r2 = 0x4000 --> r2 will be used as a mask to set bit 14
ORR r1, r1, r2            // r1 --> bitwise SET between r1 and r2
                          // r1 = 0x4000
STR r1, [r0]              // Store the value of r1 back into memory address 0x48000014
```

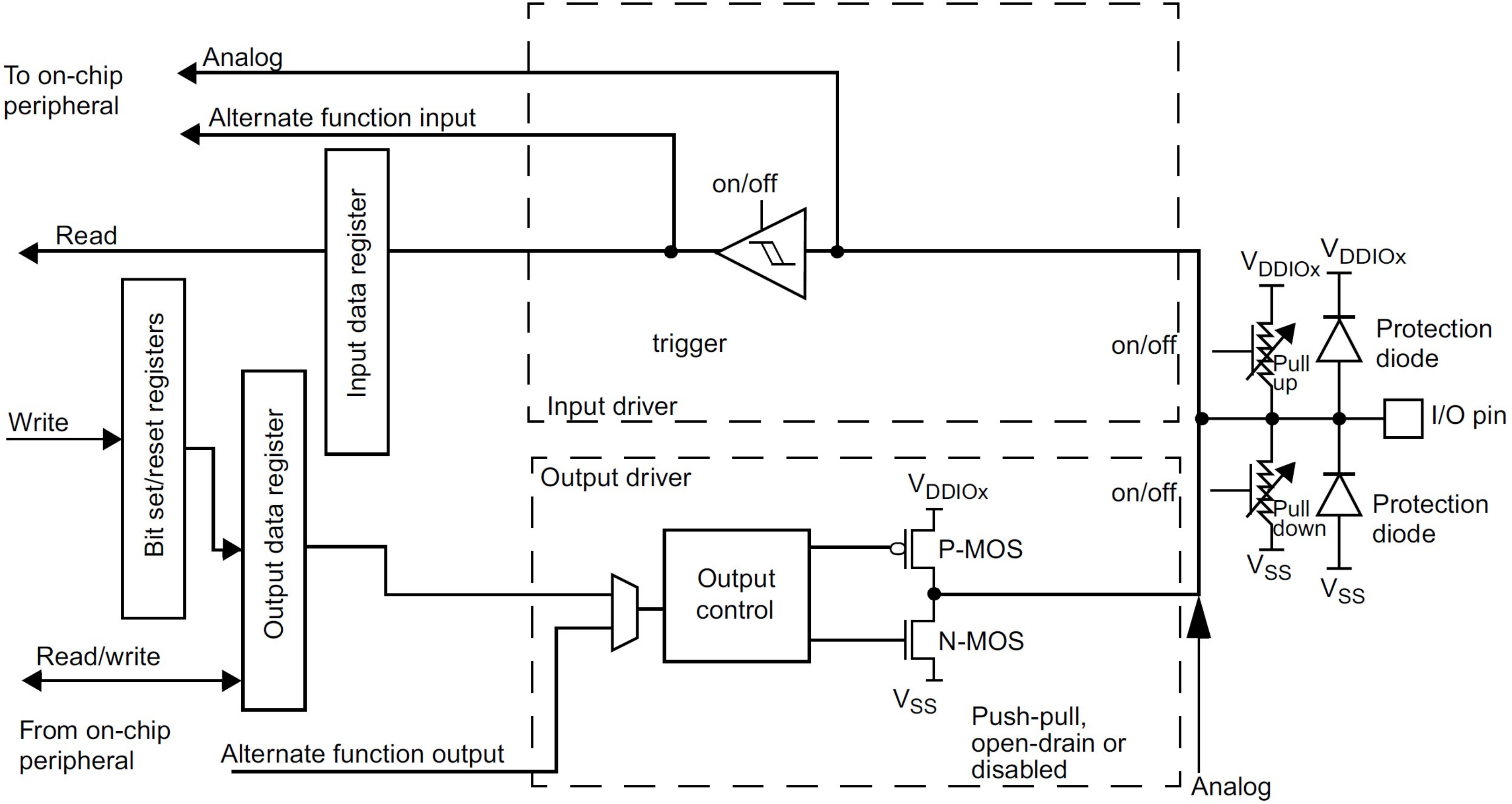
Introduction to GPIOs – Output Data Register



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

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

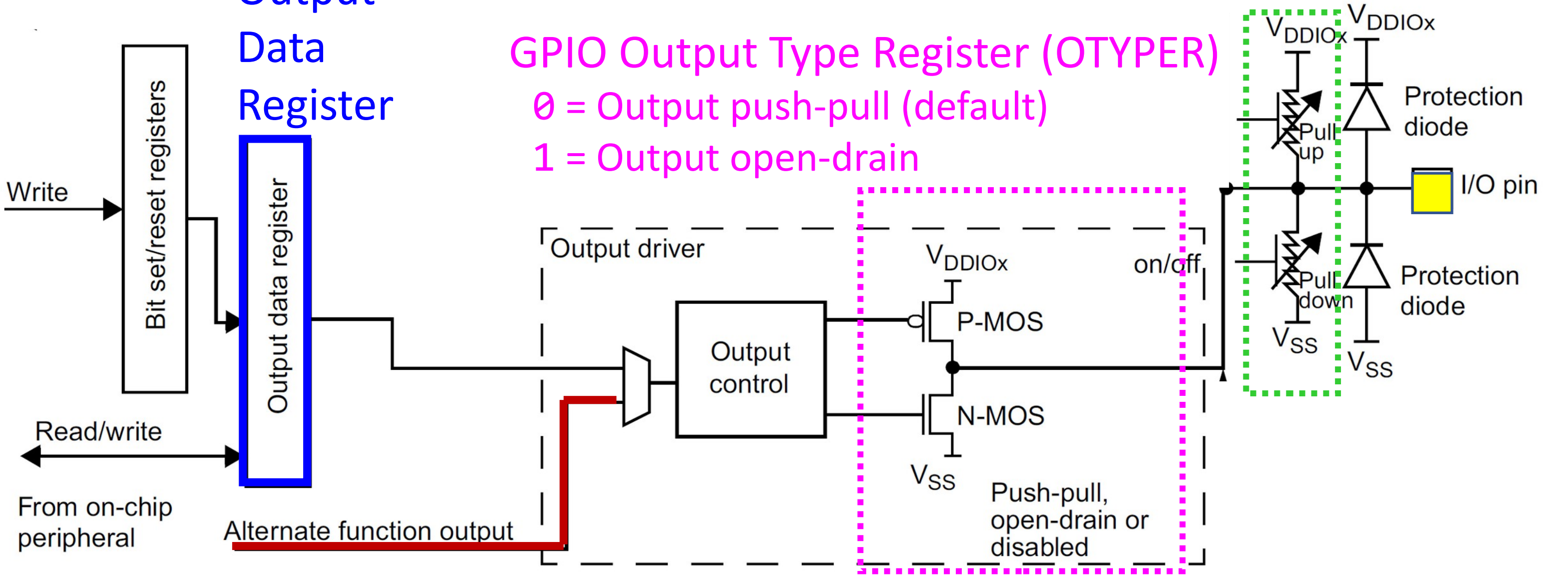
`10` = Pull-down `11` = Reserved

Output Data Register

GPIO Output Type Register (OTYPER)

`0` = Output push-pull (default)

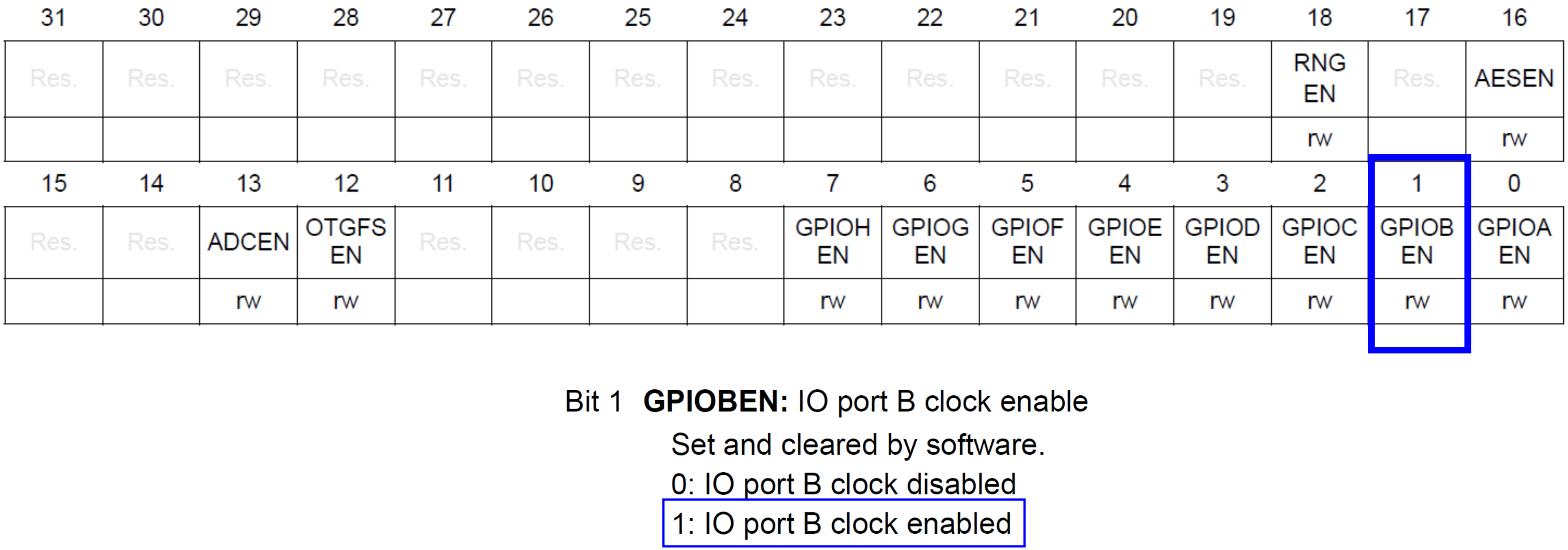
`1` = Output open-drain



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

Enabling GPIOs clocks

- **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

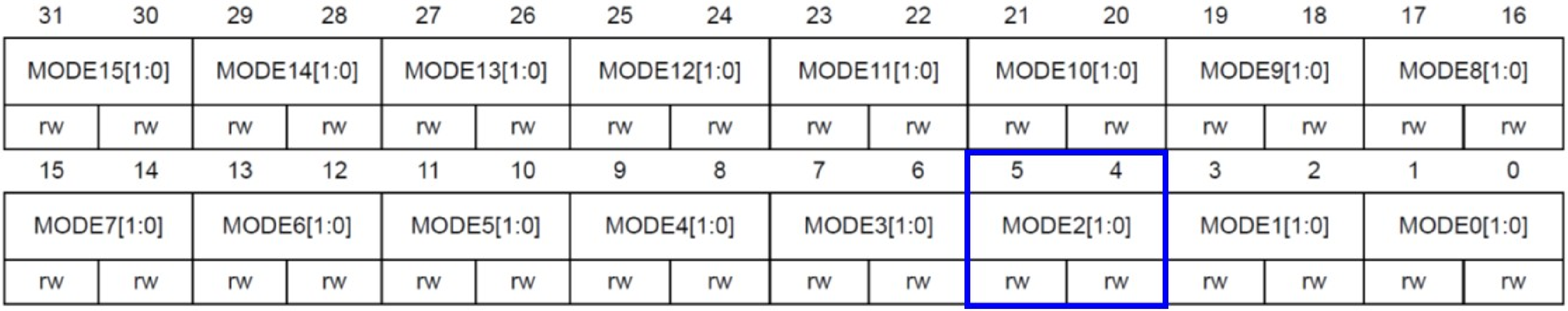
Set and cleared by software.

0: IO port B clock disabled

1: IO port B clock enabled

GPIO Mode Register (MODER)

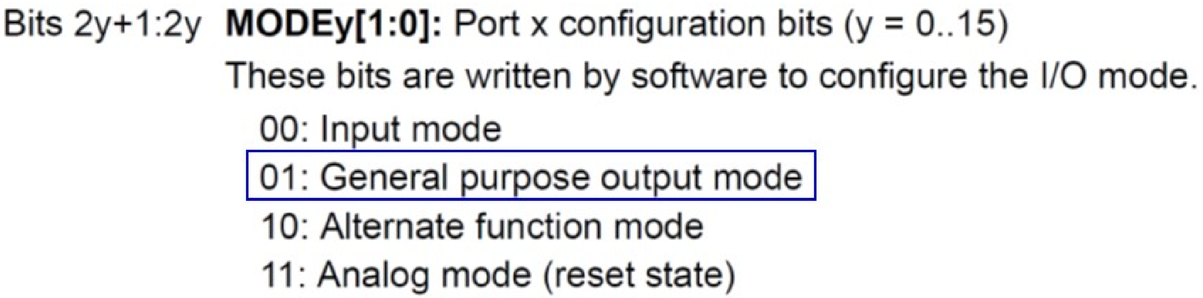
- **32 bits (16 pins, 2 bits per pin):**



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)

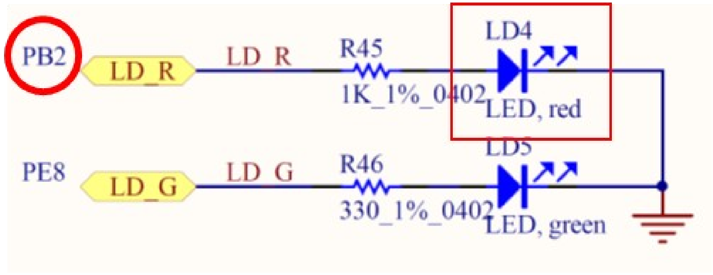
These bits are written by software to configure the I/O mode.

00: Input mode

01: General purpose output mode

10: Alternate function mode

11: Analog mode (reset state)



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.

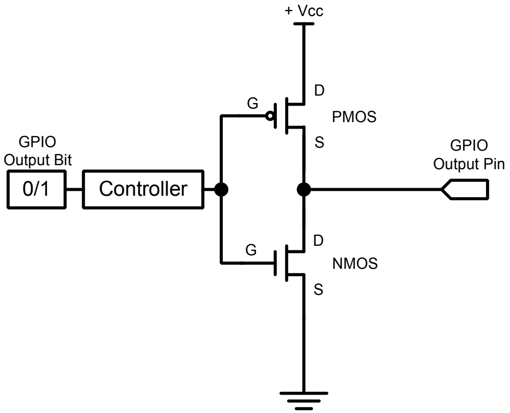
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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



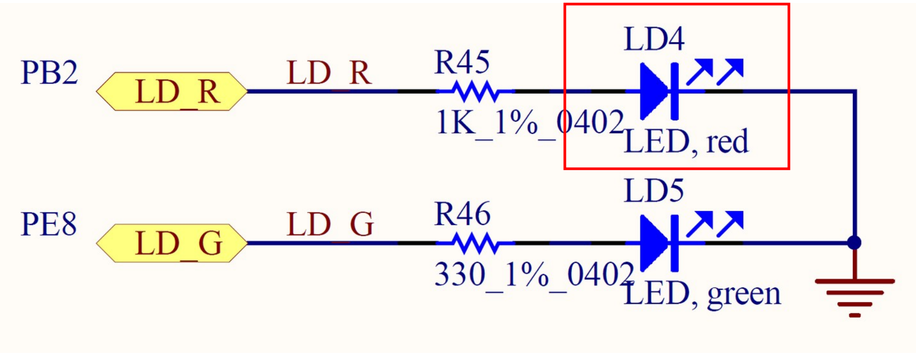
GPIO Output Data Register (ODR)

- **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



Modifying Special Purpose Registers

```
LDR r0, =#RCC_BASE
LDR r1, [r0, #RCC_AHB2ENR]
```

1st) Load the contents of the register by accessing its memory location.

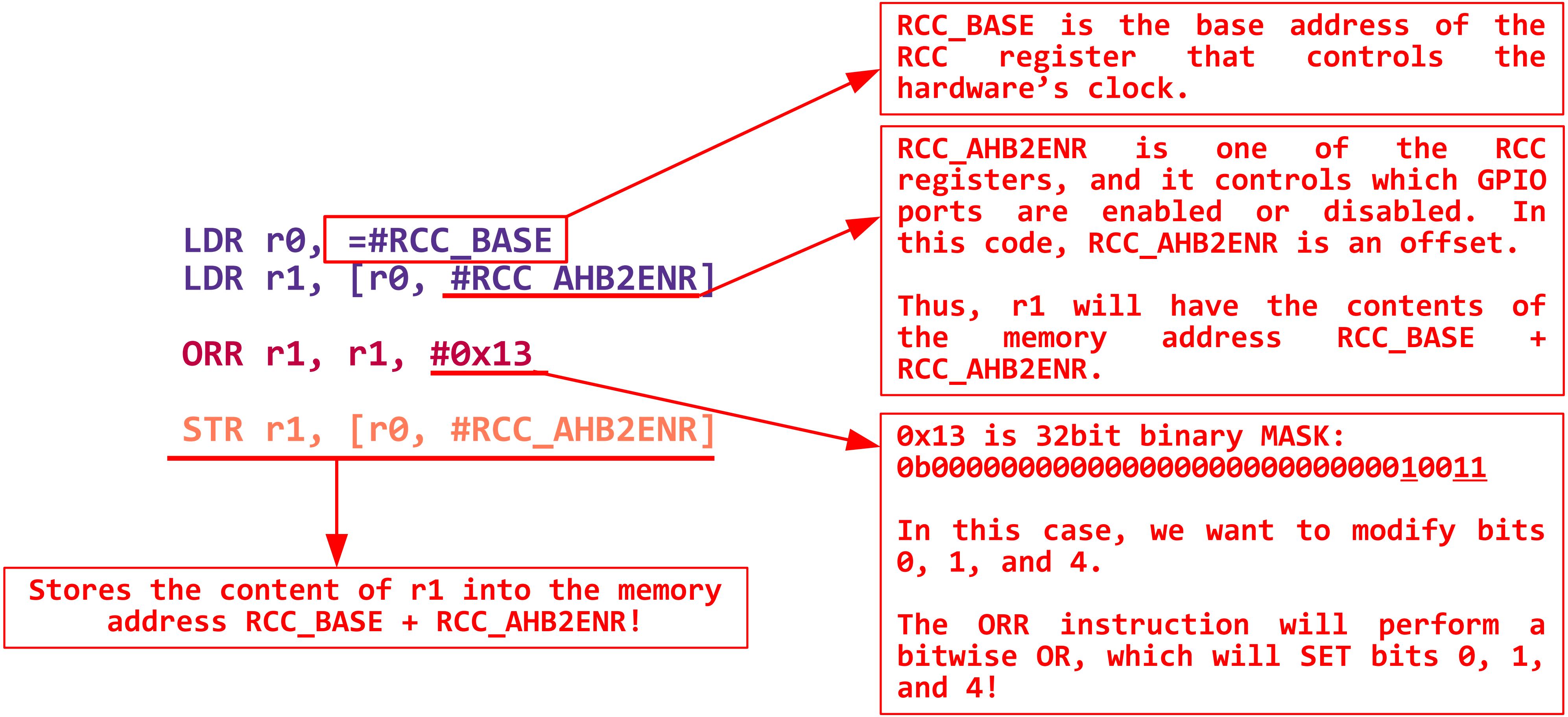
```
ORR r1, r1, #0x13
```

2nd) Modify the register’s contents.

```
STR r1, [r0, #RCC_AHB2ENR]
```

3rd) Store the modified content back to the register’s memory location.

Modifying Special Purpose Registers



```
LDR r0, =#RCC_BASE
LDR r1, [r0, #RCC_AHB2ENR]

ORR r1, r1, #0x13

STR r1, [r0, #RCC_AHB2ENR]
```

RCC_BASE is the base address of the RCC register that controls the hardware's clock.

RCC_AHB2ENR is one of the RCC registers, and it controls which GPIO ports are enabled or disabled. In this code, RCC_AHB2ENR is an offset.

Thus, r1 will have the contents of the memory address RCC_BASE + RCC_AHB2ENR.

0x13 is 32bit binary MASK:
0b000000000000000000000000000<u>1</u>00<u>11</u>

In this case, we want to modify bits 0, 1, and 4.

The ORR instruction will perform a bitwise OR, which will SET bits 0, 1, and 4!

Stores the content of r1 into the memory address RCC_BASE + RCC_AHB2ENR!

How to Read an Input - 1/2

- ***Suppose* we have a button connected to GPIO Port C, Pin 7, and we want to know if that button was pressed:**
  - **First, you have to enable GPIO Port C:**
    - `LDR r0, =#RCC_BASE`
    - `LDR r1, [r0, #RCC_AHB2ENR]`

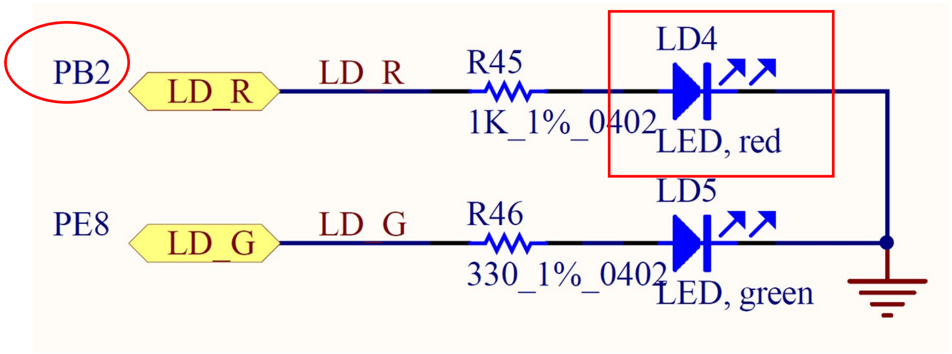
    -
      ```
      ORR r1, r1, #0x04  // 0x04 is a MASK indicating that we want to modify BIT 2. This bit enables or
                         // disables GPIO port C!
      ```
    - `STR r1, [r0, #RCC_AHB2ENR]`
  - **Second, you have to read the contents of the register `GPIOC_IDR`:**
    - `LDR r0, =#GPIOC_BASE`
    - `LDR r1, [r0, #GPIO_IDR]`

How to Read an Input - 2/2

- **Third, you have to verify if `pin 7` is equal to `1` by comparing `r1` with a mask:**
  - `AND r2, r1, #0x80 // 0x80 is equal to 0b0000000010000000 (binary)`
  - If the button is NOT pressed, `r2` will be equal to `0x00` (zero).
  - If the button IS pressed, `r2` will be equal to `0x80` (not equal to zero).
- **Fourth, you can compare r2 to zero and branch if r2 is not equal to zero:**
  - `CMP r2, #0`
  - `BNE some_label // BNE → Branch if` **`R2`** `is Not Equal to` **`ZERO`** `to the location of “some_label”`

Sample code to light up the RED LED (PB2)

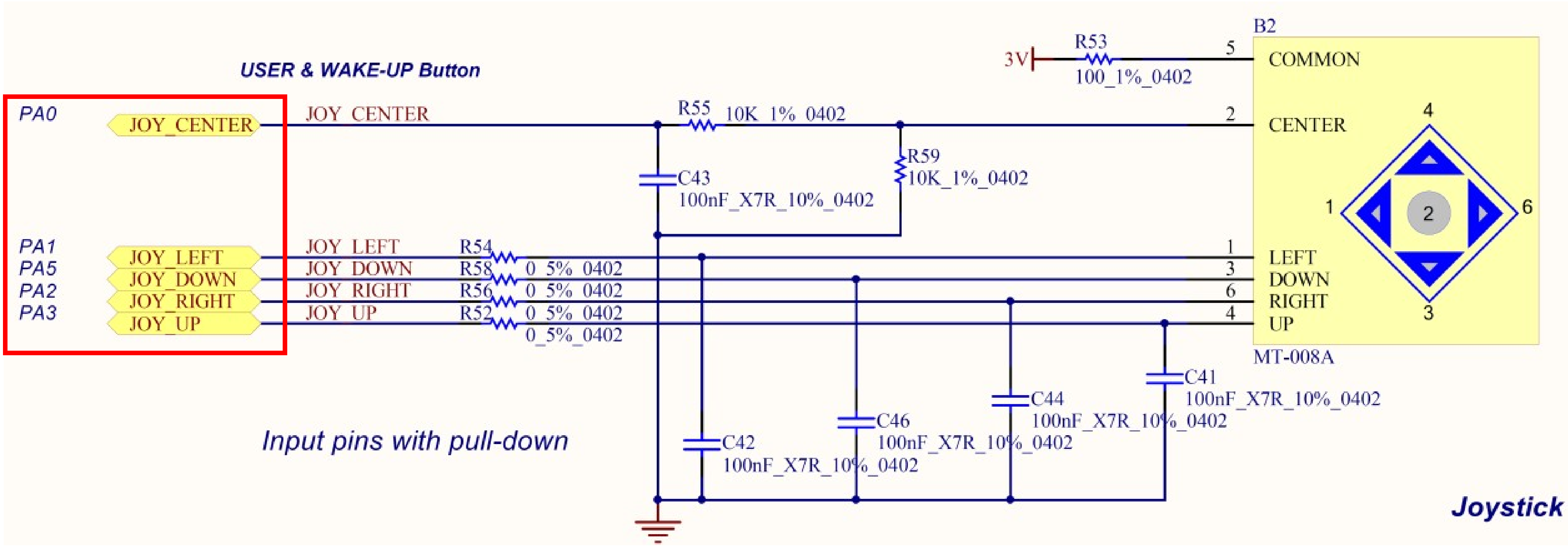
```
.syntax unified

.global main

.include "stm32l476xx_constants.s"

main:
    // Enable GPIO Port B
    LDR r0, =RCC_BASE
    LDR r1, [r0, #RCC_AHB2ENR]
    ORR r1, r1, #0x02
    STR r1, [r0, #RCC_AHB2ENR]

    // Turn ON RED LED
    LDR r0, =GPIOB_BASE
    LDR r1, [r0, #GPIO_ODR]
    ORR r1, r1, #0x04
    STR r1, [r0, #GPIO_ODR]

stop: B stop
```



These keywords are defined in the file `stm32l476xx_constants.s`! Use this file to help your while programming!

```
RCC_BASE = 0x40021000
RCC_AHB2ENR = Offset of 0x4C
GPIOB_BASE = 0x48000400
GPIO_ODR = Offset of 0x14
```

Joystick Pins



How to Create a Loop in Assembly

```
Loop:
    // DO SOME STUFF
    // DO OTHER STUFF
    B Loop  // B → Unconditional branch, jump to Loop.
```

Lab 2: step-by-step

1) Enable the GPIOs ports 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 PA0, PA1, PA2, PA3 and PA5 as **input**.

6) Configure PA0, PA1, PA2, PA3 and PA5 as **Pull-down**.

7) Wait and verify if any joystick position is pressed.

Lab 2: Start-up Code

- To help you, a start-up code is available on Canvas. Use it to create your project from scratch.
  - You will need the **main.s** and **stm32l476xx_constants.s** files in yours Src folder!
- The start-up code contains some helpful comments. Read them!