GPIO Device Driver

Lecture 07

Josh Brake
Harvey Mudd College

Outline

- Writing a simple device driver: GPIO
  - Finding information in documentation
  - Writing code to properly configure the peripheral

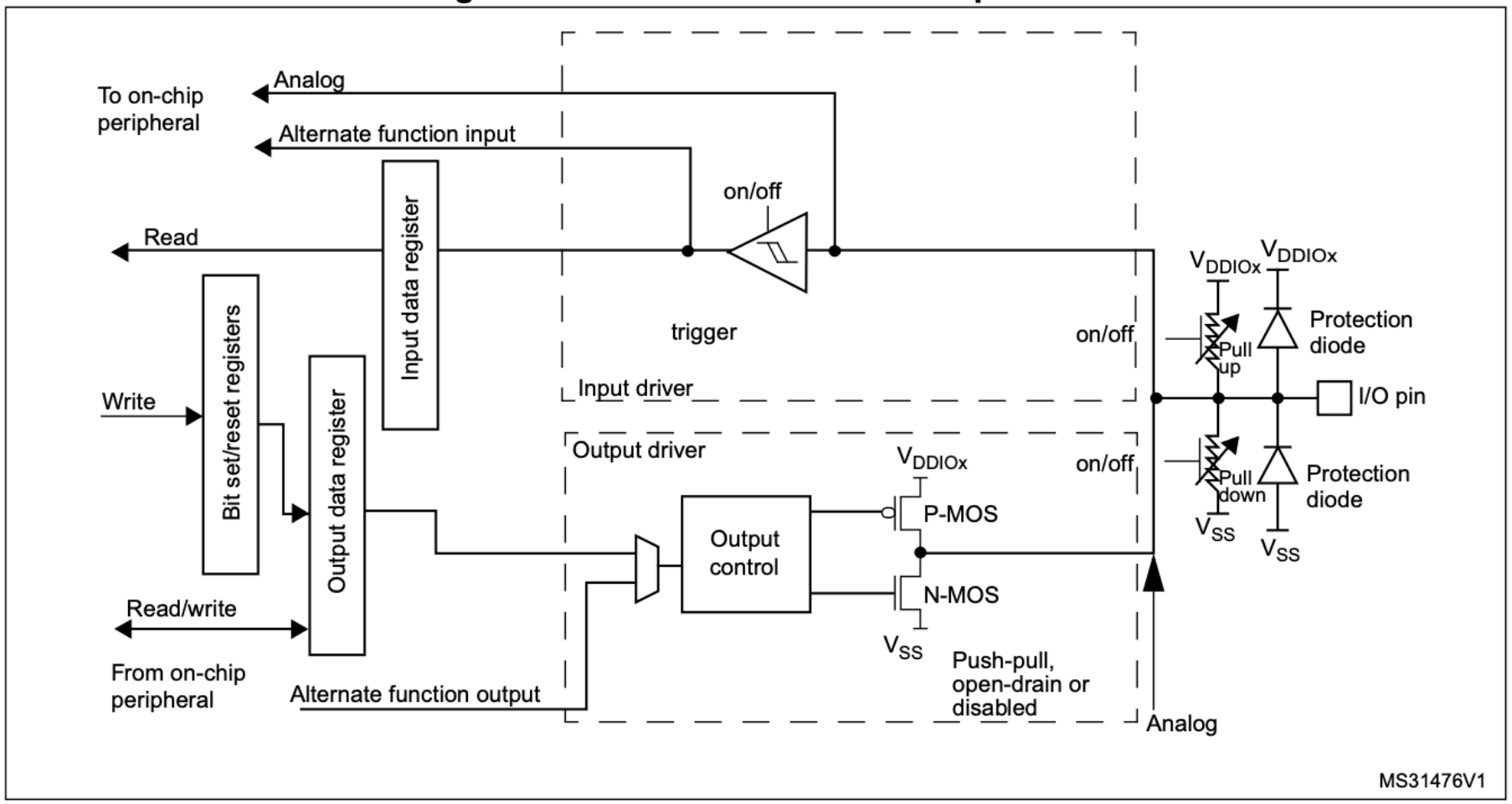
Learning Objectives

By the end of this lecture you should be able to…

- Implement basic C programming idioms and concepts (e.g., pointers, arrays, structures).
- Write a simple device driver to control the peripherals in your MCU using memory-mapped I/O.

GPIO Block Diagram

Figure 19. Basic structure of an I/O port bit



RM0394 p. 259

GPIO Register Map

Table 37. GPIO register map and reset values

Offset	Register name	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**	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	0	1	0	1	1	1	1	1	1
0x00	**GPIOx_MODER** (where x = **C..E,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..E,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

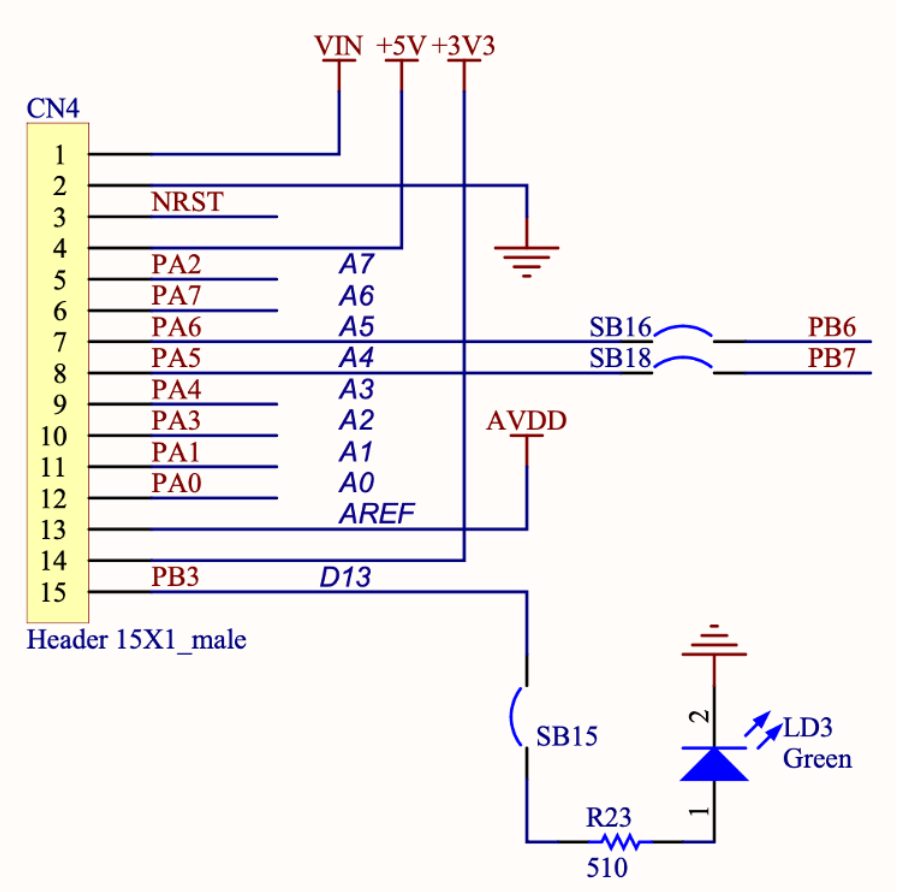
RM0394 p. 274

Steps for writing a device driver for a new peripheral

1. Look at block diagram
2. Note what elements in the diagram need to be configured
3. Find relevant registers and bits
4. Write code
    1. Base address for peripheral
    2. Create structure to define registers

Blink LED

On-board LED connected to pin PB3.



UM1956 p. 33

Enabling peripheral clock

6.2.18 Peripheral clock enable register (RCC_AHBxENR, RCC_APBxENRy)

Each peripheral clock can be enabled by the xxxxEN bit of the RCC_AHBxENR, RCC_APBxENRy registers.

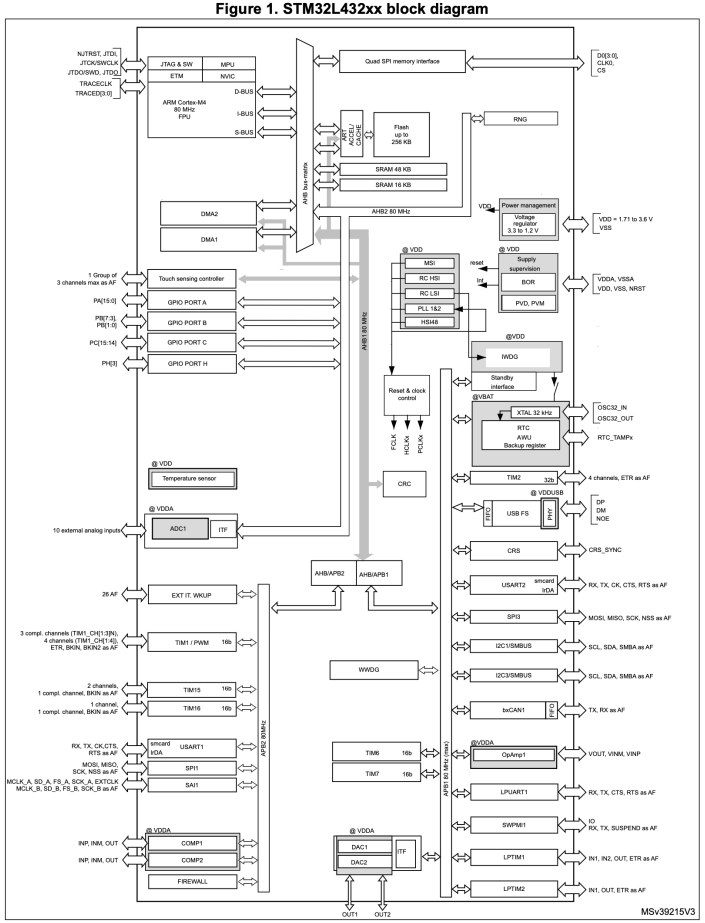
When the peripheral clock is not active, the peripheral registers read or write accesses are not supported.

The enable bit has a synchronization mechanism to create a glitch free clock for the peripheral. After the enable bit is set, there is a 2 clock cycles delay before the clock be active.

Caution: Just after enabling the clock for a peripheral, software must wait for a delay before accessing the peripheral registers.

RM0394 p. 191

Figure 1. STM32L432xx block diagram



Note: AF: alternate function on I/O pins.

DS11451 p. 13

Finding clock enable bit for GPIOB

6.4.16 AHB2 peripheral clock enable register (RCC_AHB2ENR)

Address offset: 0x4C

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access

Note: When the peripheral clock is not active, the peripheral registers read or write access is not supported.

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 (1)
													rw		rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	res.	ADCEN	res.	Res.	Res.	Res.	res.	GPIOH EN	res.	res.	GPIOE EN	GPIOD EN	GPIOC EN	GPIOB EN	GPIOA EN
		rw						rw			rw	rw	rw	rw	rw

1. Available on STM32L42xxx, STM32L44xxx and STM32L46xxx devices only.

RM0394 p. 218

Where is that bit located?

1. Look in system and memory overview section
2. Look RCC register mapping to find register and bit offsets

AHB1	0x4002 4000 - 0x4002 43FF	1 KB	TSC	*Section 23.6.11: TSC register map*
	0x4002 3400 - 0x4002 3FFF	1 KB	Reserved	-
	0x4002 3000 - 0x4002 33FF	1 KB	CRC	*Section 14.4.6: CRC register map*
	0x4002 2400 - 0x4002 2FFF	3 KB	Reserved	-
	0x4002 2000 - 0x4002 23FF	1 KB	FLASH registers	*Section 3.7.13: FLASH register map*
	0x4002 1400 - 0x4002 1FFF	3 KB	Reserved	-
	0x4002 1000 - 0x4002 13FF	1 KB	RCC	*Section 6.4.32: RCC register map*
	0x4002 0800 - 0x4002 0FFF	2 KB	Reserved	-
	4002 07FF	1 KB	DMA2	*Section 11.6.8: DMA register map and reset values*
	4002 03FF	1 KB	DMA1	*Section 11.6.8: DMA register map and reset values*
	4001 FFFF	39 KB	Reserved	

0x40	RCC_ APB2RSTR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DFSDM1RST	Res.	Res.	Res.	SAI1RST	Res.	Res.	Res.	Res.	TIM16RST	TIM15RST	Res.	USART1RST	Res.	SPI1RST	TIM1RST	SDMMC1RST	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SYSCFGRST
	Reset value								0				0					0	0		0		0	0	0								0
0x48	RCC_AHB1 ENR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TSCEN	Res.	Res.	Res.	CRCEN	Res.	Res.	Res.	FLASHEN	Res.	Res.	Res.	Res.	Res.	DMA2EN	DMA1EN
	Reset value																		0				0				1						0	0
0x4C	RCC_AHB2 ENR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RNGEN	Res.	AESEN	Res.	Res.	ADCEN		Res.	Res.	Res.	Res.	GPIOHEN	Res.	Res.	Res.	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
	Reset value																0		0			0						0				0	0	0	0	0

Configuration steps to enable basic GPIO: RCC

Turn on clock domain in Reset and Clock Control (RCC)

`RCC` base address: _____

`RCC_AHB2ENR` register offset: _____

Bit for `GPIOB_EN`: _____

Configure pin as output in GPIO register block

Configure pin as an output (GPIO_MODER)

Base address of `GPIOB`: _____

Offset of MODER register: _____

Bits in MODER to be set: _____

Value for relevant bits to configure pin as output: _____

Blink Demo: Includes

```
1  // Nucleo-L432KC Blink demo
2  // Josh Brake
3  // jbrake@hmc.edu
4  // 9/21/22
5  #include <stdint.h>
6  #define GPIOB_BASE_ADR (0x48000400UL)
7  #define RCC_BASE_ADR (0x40021000UL)
8  #define RCC_AHB2ENR ((uint32_t *) (RCC_BASE_ADR + 0x4C))
9  #define GPIOB_MODER ((uint32_t *) (GPIOB_BASE_ADR + 0x00))
10 #define GPIOB_ODR ((uint32_t *) (GPIOB_BASE_ADR + 0x14))
11 #define DUMMY_DELAY 100000
12
13 ...
```

Blink Demo: `main`

```
...

int main(void) {
  // Initialization code
  *RCC_AHB2ENR |= (1 << 1);
  // Set PB3 as output (MODER bit 7 to 0 and bit 6 to 1)
  *GPIOB_MODER &= ~(1 << 7);
  *GPIOB_MODER |= (1 << 6);
while(1) {
  for(volatile int i = 0; i < DUMMY_DELAY; i++);
  *GPIOB_ODR ^= (1 << 3);
}
```

Miscellanous Notes

Using MCU while connected to development board

- Make sure that you have the `MCU_+5V` header connected. This ensures the on-board voltage regulators work which makes sure the reset signal is held high. If not, you won't be able to connect to your MCU to program it (the reset pin will float and the MCU will always be in reset!)
- Remove jumper that came installed by default on the Nucleo board (connects reset to ground!)

Using structures to model memory-mapped I/O

```
1  // Base addresses for GPIO ports
2  #define GPIOA_BASE (0x48000000U)
3  typedef struct
4  {
5  __IO uint32_t MODER;    /*!< GPIO port mode register, Address offset: 0x00 */
6  __IO uint32_t OTYPER;   /*!< GPIO port output type register, Address offset: 0x04 */
7  __IO uint32_t OSPEEDR;  /*!< GPIO port output speed register, Address offset: 0x08 */
8  __IO uint32_t PUPDR;    /*!< GPIO port pull-up/pull-down register, Address offset: 0x0C */
9  __IO uint32_t IDR;  /*!< GPIO port input data register, Address offset: 0x10 */
10 __IO uint32_t ODR;  /*!< GPIO port output data register, Address offset: 0x14 */
11 __IO uint32_t BSRR;     /*!< GPIO port bit set/reset register, Address offset: 0x18 */
12 __IO uint32_t LCKR;     /*!< GPIO port configuration lock register, Address offset: 0x1C */
13 __IO uint32_t AFR[2];   /*!< GPIO alternate function registers, Address offset: 0x20-0x24 */
14 __IO uint32_t BRR;  /*!< GPIO Bit Reset register, Address offset: 0x28 */
15 } GPIO_TypeDef;
```

`__IO` is defined with a `#define` statement to one of the C keywords we discussed earlier. Which one?