1. Please refer to the Cortex M-4 Technical Reference Manual and the STM32F4 Discovery Reference Manual (attached to this HW) to answer this question.
(a) Section 3.4 in the Cortex M-4 Technical Reference Manual defines the memory map for this processor. What range of memory addresses is reserved for use by peripherals?

Solution:-
0x40000000 to 0x60000000

(b) Section 2.3 of the STM32F4 Discovery Reference Manual further defines the memory used by each individual peripheral. The GPIO peripherals are listed by port (GPIOA, GPIOB, GPIOC, GPIOD, etc.). What is the address range used by the GPIOD port?

Solution:-

0x4002 0C00 - 0x4002 0FFF

(c) On the STM32F4 Discovery board, each GPIO pin gets some dedicated configuration and data registers. These are described in Section 8, and especially Section 8.4, of the STM32F4 Discovery Technical Reference Manual. Please answer these questions for the following registers: MODER , OTYPER , OSPEEDR , PUPDR , IDR , ODR , and BSRR . For the BSRR register, you should further subdivide into BSRRL (BSRR low, used for bit set) and BSRRH (BSRR high, used for bit reset).
- The functionality of the register; what is it used for (in one sentence)?
- How much memory is used by this register for the entire port (typically 32 bits - 4 bytes - or 16 bits - 2 bytes)? How much memory is used for each individual pin (typically 2 bits or 1 bit)?
- What is the address offset of this register? The address offset, together with the port address, tells us exactly where in memory the register is located. For example, if the address offset of a register is 0x02 and the port uses the memory range 0x40022800 - 0x40022BFF , then the register is at address 0x40022800 + 0x02 = 0x40022802 .

Solution:-

MODERy
[1:0]: Port x configuration bits (y = 0..15).
These bits configure I/O direction mode.
Address offset:- 0x00
Sequence Meaning:-
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

GPIOx_OTYPER:- port output type register
These bits are written by software to configure the output type of the I/O port.
Reserved must be kept at reset value.
Address offset:- 0x04
Sequence Meaning:-
0: Output push-pull (reset state)
1: Output open-drain

This study source was downloaded by 100000892086212 from CourseHero.com on 10-16-2024 17:48:47 GMT -05:00

https://www.coursehero.com/file/46575646/kp2377-HW3-RTESdocx/

GPIOx_OSPEEDR:- output speed register

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

Address offset:- 0x08

Sequence Meaning:-

00: Low speed

01: Medium speed

10: High speed

11: Very high speed

GPIOx_PUPDR:- pull-up/pull-down register

These bits are written by software to configure the I/O pull-up or pull-down

Address offset:- 0x0c

Sequence Meaning:-

00: No pull-up, pull-down

01: Pull-up

10: Pull-down

11: Reserved

GPIOx_ODR:- output data register

These bits can be read and written by software

Note: For atomic bit set/reset, the ODR bits can be individually set and reset by writing to the GPIOx_BSRR register (x = A..I/J/K).

GPIOx_IDR:- input data register

These bits are read-only and can be accessed in word mode only. They contain the input value of the corresponding I/O port.

Address offset:- 0x14

GPIOx_BSRR:- bit set/reset register

BSRRL=>Bit 15:0 => BS15:BS0 => These bits are write-only and can be accessed in word, half-word or byte mode. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit

1: Sets the corresponding ODRx bit

Address offset:- 0x18

BSSRH=>Bit 31:16 => BR15:BR0 => These bits are write-only and can be accessed in word, half-word or byte mode.

A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit

1: resets the corresponding ODRx bit

Address offset:- 0x1a

This study source was downloaded by 100000892086212 from CourseHero.com on 10-16-2024 17:48:47 GMT -05:00

https://www.coursehero.com/file/46575646/kp2377-HW3-RTESdocx/

GPIOx_LCKR:- port configuration lock register

Used to lock the port configuration when a correct sequence is applied.

Address offset: 0x1C

2. This is a continuation of the previous question. Now that we know exactly where in memory each register is located, we can write C code to access these memory locations.
When you define a structure in C, the memory for elements of the structure will be laid out in the same order in which you defined them. Given that the registers are located sequentially in memory, we can use a structure in C to map registers to human readable names.

(a) We will use a GPIO_TypeDef struct to map memory for GPIO registers to human readable names. Here's a template, showing the struct definition and also how we define GPIOD using the struct:

```
typedef struct
{
...
} GPIO_TypeDef;

#define GPIOD_BASE (0xAAAAAAAA) // insert correct memory address here
#define GPIOD ((GPIO_TypeDef *) GPIOD_BASE)
```

Solution:-

Inside the typedef, we will add a line for each register that defines its total size for the entire port (32 bits - uint32_t - or 16 bits - uint16_t), its name, and a comment giving its functionality. In order to make sure the struct element is located at the correct memory offset, we need to make sure to define them in order. We will also define each element as volatile, to signal to the compiler that they may be modified outside of our code (e.g., by peripherals) and shouldn't be optimized away.
For example, the MODER register occupies 32 bits, and is located at an address offset of 0x00 (see 8.4.1 of the Technical Reference Manual). So we have to define it as the first element of the structure, and as a uint32_t. After adding MODER, our typedef would look like this:

```
typedef struct
{
        volatile uint32_t MODER; // set port mode, e.g. input, output
        ...
} GPIO_TypeDef;
```

(b) Please fill out the following C template. Add an element to the structure for each of: MODER , OTYPER , OSPEEDR , PUPDR , IDR , ODR , BSRRL , and BSRRH . Also, set the correct value for the GPIOD_BASE address based on your answer to question 1(b).

solution:-

This study source was downloaded by 100000892086212 from CourseHero.com on 10-16-2024 17:48:47 GMT -05:00

https://www.coursehero.com/file/46575646/kp2377-HW3-RTESdocx/

```
typedef struct
{
        volatile uint32_t MODER; // mode register, offset: 0x00
        volatile uint32_t OTYPER; // output type register, offset: 0x04
        volatile uint32_t OSPEEDR; // output speed register, offset: 0x08
        volatile uint32_t PUPDR; // pull-up/pull-down register, offset: 0x0C
        volatile uint32_t IDR; // input data register, offset: 0x10
        volatile uint32_t ODR; // output data register, offset: 0x14
        volatile uint16_t BSRRL; // bit set/reset low register, offset: 0x18
        volatile uint16_t BSRRH; // bit set/reset high register, offset: 0x1A
} GPIO_TypeDef;

#define GPIOD_BASE      (0x40020C00)
#define GPIOD           ((GPIO_TypeDef *) GPIOD_BASE)
```

Note that the elements must be defined in the right order and be the right sizes so that they will be located at the right memory location (based on the offset defined in the datasheet). For example, given the definition above:

- The struct begins at the memory location defined with GPIOD_BASE, 0x40020C00.
- The first element in the struct is at that memory location + 0x00 offset. So when we write to GPIOD->MODER, we are writing to the memory location 0x40020C00.
- The second element in the struct (OTYPER) is at 0x40020C00 + the additional bytes used by the MODER register. Since MODER is defined to use 32 bits (4 bytes), OTYPER will be at 0x40020C00 + 0x04 offset = 0x40020C04
- The third element in the struct (PUPDR) is at 0x40020C00 + the additional bytes used by the previous elements, the MODER and OTYPER registers. Since MODER and OTYPER are both defined to use 32 bits (4 bytes), 8 bytes total, PUPDR will be at 0x40020C00 + 0x08 offset = 0x40020C08
- Following a similar procedure, you can see that all of the struct elements will be located at the correct memory addresses. (Note that some elements are defined as 16 bits).

If you look in stm32f4xx.h you'll see a similar struct definition. So now we've learned how to write a basic GPIO peripheral driver, using just the datasheet for reference.

(c) Which element(s) of the struct do you think you might declare as a volatile const instead of just a volatile ? Why?

We might declare IDR as volatile const because it should be read-only to us (we never write to an input data register).

(d) Do we need to use malloc() in C to allocate memory for GPIOD? Why or why not?

Solution: No, we do not need to use malloc() to allocate memory for GPIOD (and shouldn't). malloc() is used to allocate memory dynamically on the heap (this memory will be in SRAM, which is located at memory addresses starting at 0x20000000).

We are using memory in the peripherals address range (starting at 0x40000000). We don't want

This study source was downloaded by 100000892086212 from CourseHero.com on 10-16-2024 17:48:47 GMT -05:00

https://www.coursehero.com/file/46575646/kp2377-HW3-RTESdocx/

memory on the heap/in SRAM. We explicitly define our pointers to use the correct memory addresses, in the peripherals range.

(e) Please answer these questions for the following registers: MODER , OTYPER , OSPEEDR , PUPDR , IDR , ODR ,
BSRRL , and BSRRH :

- For all registers except IDR , write a line of C code that assigns the value '1' to this register for pin 3 of the GPIOD port, without affecting the values set for any other pins. For examples, see pages 12 and 14 of the lecture slides on peripherals.
- For all registers except IDR , write a line of C code that assigns the value '1' to this register for all pins on the GPIOD port.
- What does setting a value of '1' do on this register?

Solution: To set values only for pin 3:

```
// To reliably set '01' in a *two-bit* register, we have to
// clear the bits (or at least, the bit on the left)
// and then set the bit on the right to 1

// MODER has two bits per pin
GPIOD->MODER &= ~(3 << ((2*3)+1));
GPIOD->MODER |= 1 << 2*3;

// OTYPER has one bit per pin
GPIOD->OTYPER |= 1 << 3;

// OSPEEDR has two bits per pin
GPIOD->OSPEEDR &= ~(3 << ((2*3)+1));
GPIOD->OSPEEDR |= 1 << 2*3;

// PUPDR has two bits per pin
GPIOD->PUPDR &= ~(3 << ((2*3)+1));
GPIOD->PUPDR |= 1 << 2*3;

// ODR has one bit per pin
GPIOD->ODR |= 1 << 3;


// BSRRL and BSRRH have one bit each per pin
// Note that for the BSRR registers, we don't really need
// to use |= - for these registers, we can just use =
// and the result will be the same
GPIOD->BSRRL |= 1 << 3;
GPIOD->BSRRH |= 1 << 3;

//To set values for all pins on the port:
// MODER has two bits per pin
GPIOD->MODER = 0x55555555;

// OTYPER has one bit per pin, and we shouldn't touch
// the reserved bits - just set lower 16 bits
GPIOD->OTYPER |= 0xffff;

// OSPEEDR has two bits per pin
GPIOD->OSPEEDR = 0x55555555;
```

This study source was downloaded by 100000892086212 from CourseHero.com on 10-16-2024 17:48:47 GMT -05:00

https://www.coursehero.com/file/46575646/kp2377-HW3-RTESdocx/

```
// PUPDR has two bits per pin
GPIOD->PUPDR = 0x55555555;

// ODR has one bit per pin, and we shouldn't touch
// the reserved bits - just set lower 16 bits
GPIOD->ODR |= 0xffff;

// BSRRL and BSRRH have one bit each per pin
GPIOD->BSRRL = 0xffff;
GPIOD->BSRRH = 0xffff;
```

Setting these to 1 has the following effect:

- MODER: Sets pin(s) to general purpose output mode
- OTYPER: Sets output pin(s) to open drain type
- OSPEEDR: Sets speed to medium speed
- PUPDR: Sets a pull-up resistor on pin(s)
- ODR: Writes a high value to output pin(s)
- BSRRL: Sets a value of 1 on output pin(s)
- BSRRH: Resets (sets a value of 0) on output pin(s)

This study source was downloaded by 100000892086212 from CourseHero.com on 10-16-2024 17:48:47 GMT -05:00

https://www.coursehero.com/file/46575646/kp2377-HW3-RTESdocx/