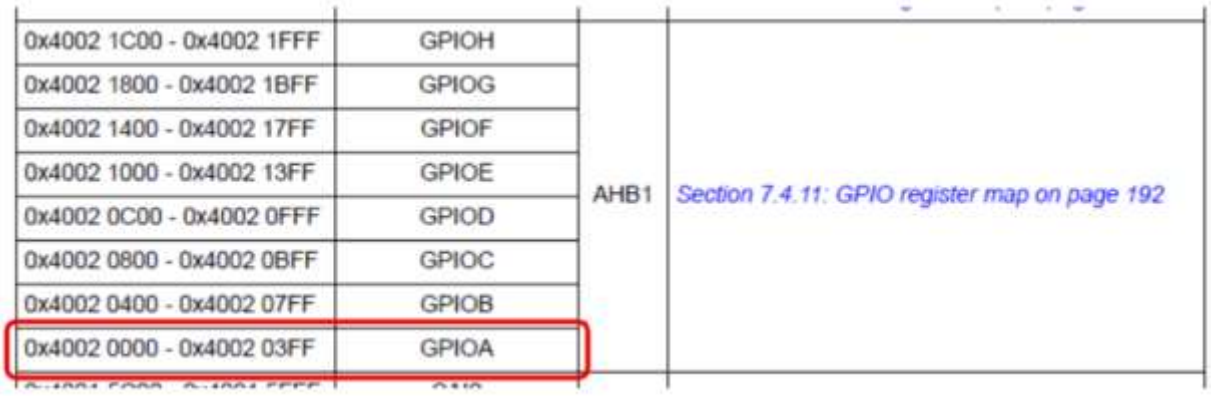
EE 3314 HW2: Baremetal GPIO Programming

Write lines of embedded code for each the following GPIO Use Cases. Follow the convention from the examples in the lectures. For each case, you will need to:

- Declare and initialize pointer variables for accessing the needed registers for configuring and accessing GPIO peripheral pins. Don't forget a pointer for enabling the peripheral clock!
- Use const and volatile in your pointer declarations
- Use the pointers to enable to peripheral clock, set the GPIO mode for the pin, and read or write data to the GPIO pin as described



0x4002 1C00 - 0x4002 1FFF	GPIOH	AHB1	Section 7.4.11: GPIO register map on page 192
0x4002 1800 - 0x4002 1BFF	GPIOG		
0x4002 1400 - 0x4002 17FF	GPIOF		
0x4002 1000 - 0x4002 13FF	GPIOE		
0x4002 0C00 - 0x4002 0FFF	GPIOD		
0x4002 0800 - 0x4002 0BFF	GPIOC		
0x4002 0400 - 0x4002 07FF	GPIOB		
0x4002 0000 - 0x4002 03FF	GPIOA		

#define RCC_AHB1ENR 0x40023830

#define GPIOA_Base 0x40020000

#define GPIOB_Base 0x40020400

#define GPIOC_Base 0x40020800

#define GPIOD_Base 0x40020C00

#define MODER_OFFSET 0x00

#define INPUT_IDR 0x10

#define OUTPUT_ODR 0x14

1. Use GPIOB Pin 4 as an output. Set the state of the pin to a logic high

const

```
volatile uint32_t *pRCC_AHB1ENR =   (volatile uint32_t*) (RCC_AHB1ENR + MODER_OFFSET);
   volatile uint32_t *pGPIOB_Moder = (volatile uint32_t*) (GPIOB_Base + MODER_OFFSET);
   volatile uint32_t *pGPIOB_ODR   = (volatile uint32_t*) (GPIOB_Base + OUTPUT_ODR);

//enables bit 1
  *pRCC_AHB1ENR |= (1 << 1);          //what page/chapter?

  //sets PB4 as output, Pin 4 is 8 and 9on 7.4.1, need (01)
  *pGPIOB_Moder &= ~(3 << (4 * 2));
  *pGPIOB_Moder != ~(1 << (4 * 2));

  //set PB4 High, on 7.4.6
  *pGPIOB_ODR |= (1 << 4);
```

2. Use GPIOC Pin 9 as in input. Read the state of the pin into a variable “pinStatus”

```
volatile uint32_t *pRCC_AHB1ENR = (volatile uint32_t*) (RCC_AHB1ENR + MODER_OFFSET);
   volatile uint32_t *pGPIOC_Moder = (volatile uint32_t*) (GPIOC_Base + MODER_OFFSET);
   volatile uint32_t *pGPIOC_IDR   = (volatile uint32_t*) (GPIOC_Base + INPUT_IDR);
   uint32_t pinStatus;

//enables bit 1
  *pRCC_AHB1ENR |= (1 << 2);          //what page/chapter?

  //sets PC9 as input (00)

  *pGPIOC_Moder &= ~(3 << (9 * 2));

  //set PC9 as
  pinStatus = (*pGPIOC_IDR >> 9) & 0x1;
```

3. Use GPIOA Pin 15 as an output. Set the state of the pin to a logic low

```
volatile uint32_t *pRCC_AHB1ENR = (volatile uint32_t*) (RCC_AHB1ENR + MODER_OFFSET);
volatile uint32_t *pGPIOA_Moder = (volatile uint32_t*) (GPIOA_Base + MODER_OFFSET);
volatile uint32_t *pGPIOA_ODR   = (volatile uint32_t*) (GPIOA_Base + OUTPUT_ODR);


//enables bit 1
 *pRCC_AHB1ENR |= (1 << 0);

 //sets PA15 as output (01)

  *pGPIOA_Moder &= ~(3 << (15 * 2));
  *pGPIOA_Moder != ~(1 << (15 * 2));

 //set PA15 as a low
 *pGPIOA_ODR &= (1 << 15);
```

4. Use GPIOD Pin 1 as in input. Read the state of the pin into a variable “pinStatus”

```
volatile uint32_t *pRCC_AHB1ENR = (volatile uint32_t*) (RCC_AHB1ENR + MODER_OFFSET);
volatile uint32_t *pGPIOD_Moder = (volatile uint32_t*) (GPIOD_Base + MODER_OFFSET);
volatile uint32_t *pGPIOD_ODR   = (volatile uint32_t*) (GPIOD_Base + INPUT_IDR);

unsigned int pinStatus;


//enables bit 1
 *pRCC_AHB1ENR |= (1 << 3);

 //sets PD1 as input (00)

  *pGPIOD_Moder &= ~(3 << (1 * 2));

//set pinStatus as PD1
```

```
*pGPIOD_ODR &= (1 << 15);

  //set PD1 as
   pinStatus = (*pGPIOD_ODR >> 1) & 0x1;
```