NYU TANDON SCHOOL OF ENGINEERING　　　Name: Anshika Jain

EL648 – Real Time Embedded Systems　　　ID: aj3437

Midterm 1 Fall 2021

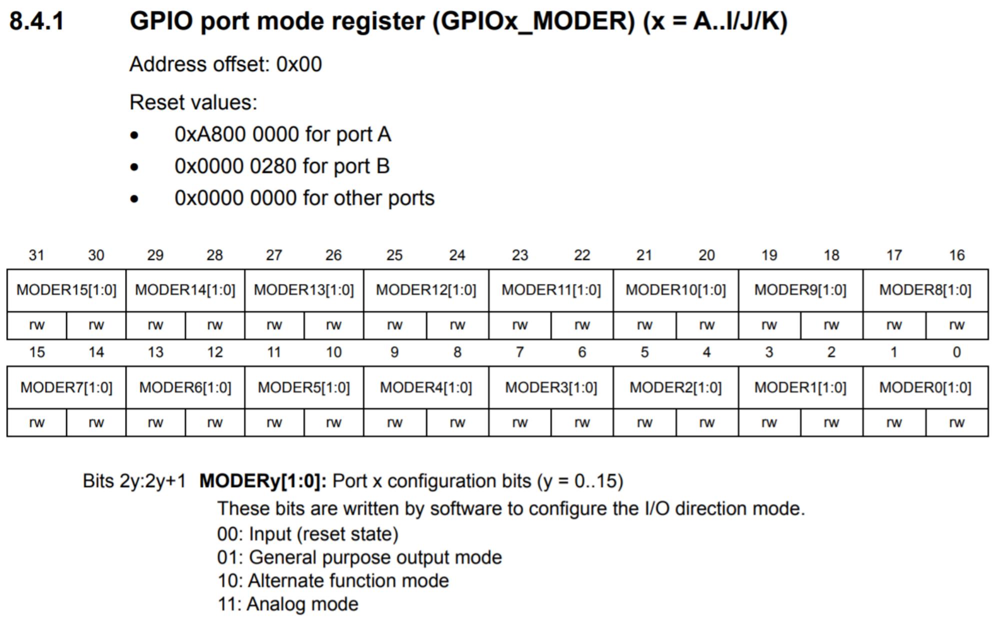
Instructions: Answer all questions on this paper. You may use any reference materials from class or from our class websites and/or reference sheets.

**Question 1:**

Although GPIO is not specifically part of this exam, we spent a significant amount of time in class discussing how to read and write memory using both C and assembly. Since our architecture uses memory mapped I/O, setting up the GPIO registers is simply reading and writing the appropriate memory locations. The following algorithm can be used for our controller to do a block GPIO write, meaning writing all port pins (15 in total) simultaneously, for any port (say PORTA).

## PORTA has a base address of 0x40020000

***A. Set each pin in the MODER register to general purpose output mode***



**8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)**

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MODER15[1:0] | | MODER14[1:0] | | MODER13[1:0] | | MODER12[1:0] | | MODER11[1:0] | | MODER10[1:0] | | MODER9[1:0] | | MODER8[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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MODER7[1:0] | | MODER6[1:0] | | MODER5[1:0] | | MODER4[1:0] | | MODER3[1:0] | | MODER2[1:0] | | MODER1[1:0] | | MODER0[1:0] | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

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

These bits are written by software to configure the I/O direction mode.
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

***B.  Set each pin in the OTYPER register to output push-pull***

### 8.4.2 GPIO port output type register (GPIOx_OTYPER) (x = A..I/J/K)

Address offset: 0x04

Reset value: 0x0000 0000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | |
| **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 31:16 Reserved, must be kept at reset value.

Bits 15:0 **OTy**: Port x configuration bits (y = 0..15)
These bits are written by software to configure the output type of the I/O port.
0: Output push-pull (reset state)
1: Output open-drain

***C.  Set each pin in the PUPDR register to 0, meaning no pull up or down resistor***

### 8.4.4 GPIO port pull-up/pull-down register (GPIOx_PUPDR) (x = A..I/J/K)

Address offset: 0x0C

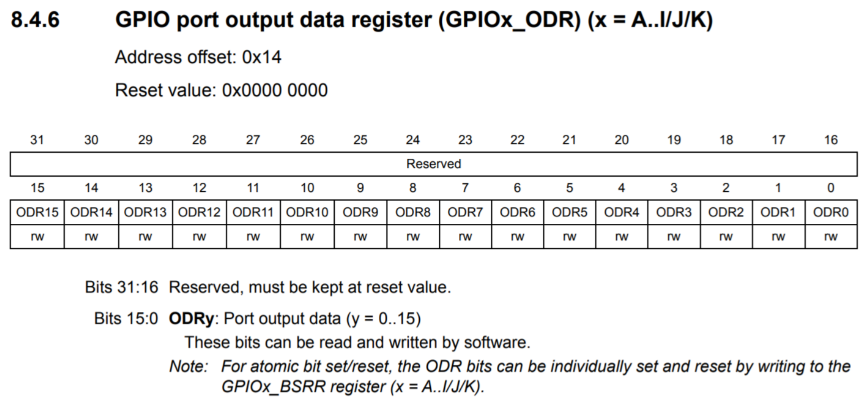
Reset values:

- 0x6400 0000 for port A
- 0x0000 0100 for port B
- 0x0000 0000 for other ports

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PUPDR15[1:0] | | PUPDR14[1:0] | | PUPDR13[1:0] | | PUPDR12[1:0] | | PUPDR11[1:0] | | PUPDR10[1:0] | | PUPDR9[1:0] | | PUPDR8[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** |
| PUPDR7[1:0] | | PUPDR6[1:0] | | PUPDR5[1:0] | | PUPDR4[1:0] | | PUPDR3[1:0] | | PUPDR2[1:0] | | PUPDR1[1:0] | | PUPDR0[1:0] | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bits 2y:2y+1 **PUPDRy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O pull-up or pull-down
00: No pull-up, pull-down
01: Pull-up
10: Pull-down
11: Reserved

***D. Set the ODR register bits to a 1 or 0, depending on if the pin is turned on or off.***



**8.4.6 GPIO port output data register (GPIOx_ODR) (x = A..I/J/K)**

Address offset: 0x14

Reset value: 0x0000 0000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | |
| **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| ODR15 | ODR14 | ODR13 | ODR12 | ODR11 | ODR10 | ODR9 | ODR8 | ODR7 | ODR6 | ODR5 | ODR4 | ODR3 | ODR2 | ODR1 | ODR0 |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy**: Port output data (y = 0..15)
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).*

**a. Write a C function:**

```
uint32_t  BlockWritePortA(uint16_t PinVals) {

//Code Here
}
```

PinVals is a 16 bit number with 1's in positions to turn "on" and 0's in the pin positions to turn "off". The function should implement algorithms parts A., B., C. and D. described above and return the value of the ODR register above. Note, the addresses of the registers (memory) are the Base Address + Offset as indicated above.

**b. Write an ARM assembly function called "_BlockWritePortA" that implements part A above.**

**NOTE: Be careful how you use LDR, as there are limits to the "mem" parameter!**

```
.globl _BlockWritePortA

.syntax unified


_BlockWritePortA

;

; Code Here

;

bx lr
```

**c. Write the short C program that calls the ARM Assembly function _BlockWritePortA in its main();**

## Question 1 Solution:

A)
for the operation the to set all pins the bits will have alterante 0 and 1 value according to 2y:2y+1

B)
all bits will be 2y:2y+1 will be 0

in the ODR register, In order to Set a pin, we can directly write a ‘1’ to the respective bit, and in order to Reset it, write a ‘0’ in the respective bit

C)
For operation all the bits will be 0 from
0-31

D)
a) given that I want to set all the pins hence to set all the pins the 2y and 2y+1 number must be set to 0:1 hence for that the function is :

```
uint32_t BlockWritePortA(uint16_t PinVals) {

pinVal = GPIOA->IDR;

pinVals->MODER l= (1<<2*x);  // where x is the pin number the STM cortex has few as 16 pins.

// to output push pull for all the pins set the bits to 0 for that pin                Text

pinVals->OTYPER &= ~(1<<x);  // bit x=0 --> Output push pull where x is the pin. number.

// to set the pin to no push up r pull down we set the 2y:2y+1 bit to 0:0

pinVals->PUPDR &= ~((1<<2x) l (1<<2x+1));  // Pin PAx are 0:0 --> no pull up or pulldown where x is pin number

// if we want the pin to be on or off hence we will set the pin according with the bit corresponding to the pin

pinVals->ODR l= 1<<x; // Set the Pin PAx. where x is the pin number

pinVals->ODR &= ~(1<<x); // Reset the Pin PAx where x is the pin number

uint16_t odrresult = GPIO_ReadInputDataBit(GPIOA);
return odrresult;

}
```

b)

```
SETgpioPins :

// configure GPIO 20 for output
//
// FSEL0 = 0x20200000 (GPIO0-GPIO9)
// FSEL1 = 0x20200004 (GPIO10-GPIO19)
// FSEL2 = 0x20200008 (GPIO20-GPIO29)
// ...
mov r0, #0x20 // r0 = 0x00000020
lsl r1, r0, #24 // r1 = 0x20000000
lsl r2, r0, #16 // r2 = 0x00200000
orr r0, r1, r2 // r0 = 0x20200000
orr r0, r0, #0x08 // r0 = 0x20200008
mov r1, #1 // r1 = 1 is OUTPUT
str r1, [r0]

bx ldr
```

We can do same for other pins as well

c)

```
extern int SETgpioPins(uint16_t val);
```

**Question 2:**

Consider the following ARM assembly code segment.

a. Accurately comment each line of code
b. Describe what parameters r0 and r1 (passed into the function) are used for.
c. What are each of the local variables r4-r8 used for?
d. What is the purpose of this function?
e. Explain in detail the specific purpose of “stmfd” and “ldmfd” in this function.

```
.globl _MyFunc
.text
_MyFunc:
 stmfd sp!, {r4, r5, r6, r7, r8, lr}   push all the resgisters into a stack whose base address is stack pointer
 cmp r1, #1       compare and subtract one for r1 using as a counter
 ble end_outer    branch to end_outer loop if it is equal to, branch if nothing to do

 sub r5, r1, #1   subtracts 1 from r1 and stores it in r5
 mov r4, r0       move the address of r0 register to r4
 mov r6, #0        move 0 to the register
                   r6 set when we swap

loop_start:       The loop starts , loop_start is the label
 ldr r7, [r4], 4    load the value of r4 into register r7 and then update the value of r4 by 4 offset
 ldr r8, [r4]     load the value of r4 into register r8
 cmp r7, r8       compare the values of r7 and r8 by subtracting them if greater
 ble no_go        branch if the label is equal to no_go branch if second greater

 mov r6, #1       move the value of 1 into the register r6
 sub r4, r4, 4    subtract 4 from r4 and store it in r4. reset the pointer to first element
 swp r8, r8, [r4]    to swap the contents of r8 with the address of r4 and store it in r8
 str r8, [r4, 4]!    r4(base register) is updated by 4 and then r8 is stored in r4. store new  value r8 to increased address
no_go:
 subs r5, r5, #1     subtract 1 from r5 and store the value in r5
 bne loop_start    if branch not equal condition is true then go to label loop_start to restart the loop if more needed

end_inner:        defining label end_inner
 cmp r6, #0        comparing value of register r6 with 0 by subtracting
 beq end_outer     if branch equal then go to label end_outer

 mov r6, #0        move the value 0 into the register r6
 mov r4, r0        move the value of register r0 into the register r4 to reset the pointer
 sub r5, r1, #1       subtracts 1 from r1 and stores it in r5,
                      to reset the counter
 b loop_start      always executed branch to loop_start again

end_outer:
        ldmfd   sp!, {r4, r5, r6, r7, r8, pc}   to pop out all the values of the registers from a full descending stack
```

## Question 2 Solution:

b) r1 is value that is used to
comapre the values in the
registers
r0 is base address of the stack

c)
the local values of r4 -r8 are used for to push the values or create memory locations for the values to bes stored in stack

d) the function is doing the operation from making the values from descending to ascending
order, generally sorting(bubble)

e)
The stack grows downwards, starting with a high address and progressing to a lower one
(a descending stack), or upwards, starting from a low address and progressing to a
higher address (an ascending stack).

The stack pointer can either point to the last item in the stack (a full stack), or the next
free space on the stack (an empty stack)

To make it easier for the programmer, stack-oriented suffixes can be used instead of the
increment or decrement, and before or after suffixes.

the stmfd stands for  to store in stack which is full in descending order and ldmfd means
the to  pop the stack contents in full descending order .

STMFD (STMDB, Decrement Before)

LDMFD (LDM, increment after)

```
  STMFD    sp!, {r0-r5}  ; Push onto a Full Descending Stack
    LDMFD    sp!, {r0-r5}  ; Pop from a Full Descending Stack
```