

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 (GPIO<sub>x</sub>\_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										
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										

Bits 2y:2y+1 **MODER<sub>y</sub>[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

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) C Program Here

```
uint32_t BlockWritePortA(uint16_t PinVals){  
    uint32_t *a;  
    uint16_t *b;  
    a = 0x40020000;  
    *a = 0x55555555;  
    a = 0x40020004;  
    *a = 0x0000 0000;  
    a = 0x4002000C;  
    *a = 0X 0000 0000;  
  
    b = 0x40020016;  
    *b = 0X 0000;  
    b = 0x40020014;  
    *b = PinVals;  
    a = 0X 4002 0014  
    return *a;  
}
```

(b) Assembly Here:-

\_BlockWritePortA =

```
    mov  r2, #40020000 ;  
    mov  r3, #55555555 ;  
    str  r3, [r2];  
    add  r2, r2, #0x04  
    mov  r3, #0X0000 0000  
    Str  r3, [r2]  
    add  r2, r2, #08  
    mov  r3, #0000 0000  
    Str  r3, [r2]  
    add  r2, r2, #10  
    Str  r0, [r2]  
    sub  r2, r2, #0X2  
    ldr  r0, [r2]  
    BX  lr;
```

(c)

```
#include <mbed.h>  
#include <string.h>  
extern"C" uint32_t BlockWritePortA(uint16_t PinVals);  
uint16_t PinVals;  
int main() {  
    Pinvals = 0xFF00; //Can be changed by requirements.  
    -BlockWritePortA(&PinVals);  
}
```

## 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.

a)

Question 2 Solution:  $r_1$  = how many numbers you want to sort / length of the array

b) if  $r_1 = 1$ , there will be no operation for others, the program ends.

c) if  $r_1 \neq 1$ ,  $r_1$  is the length of the array, do the sort operation

$r_0$  = the start address of the array. The first number of array stores in it.

c)  $r_4$ : The address that will be compared with the latter one.

$r_5$ : Use to count the number, initial value =  $r_1$ , when  $r_5 = 0$ , program ends.

$r_6$ : Use to check whether program should end inner, instead of outer,

$r_7$  &  $r_8$ :  $r_7, r_8$  stores the data to check which number inside is smaller, Put the smaller number front and put the larger one back.  $r_8$  is also a register to temp store.

d) This function is used to make the array sorted with non-decreasing order.

e)

STMFD // Put  $r_4, r_5, r_6, r_7, r_8, lr$  into stack which  
SP pointed to

LDMFD // pop  $r_4, r_5, r_6, r_7, r_8, pc$  out of stack  
which SP pointed to

These two operations are used to push all the registers we use into the stack, maintain them, and pop them out and release them.