NYU TANDON SCHOOL OF ENGINEERING Name: ________________________________

ECE6483 – Real Time Embedded Systems ID: ________________________________

EXAM 2 / FALL 2022

Instructions: Answer all questions. You may uses any reference materials from class or from our class websites and/or reference sheets.

Question 1:

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 (steps 1.a, 1.b etc.)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).

Use the below algorithm steps (<u>LOCATED ON THE FOLLOWING PAGES</u>) to 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 1.a, 1.b, 1.c and 1.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.

PORTA has a base address of 0x40020000

1.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

1.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

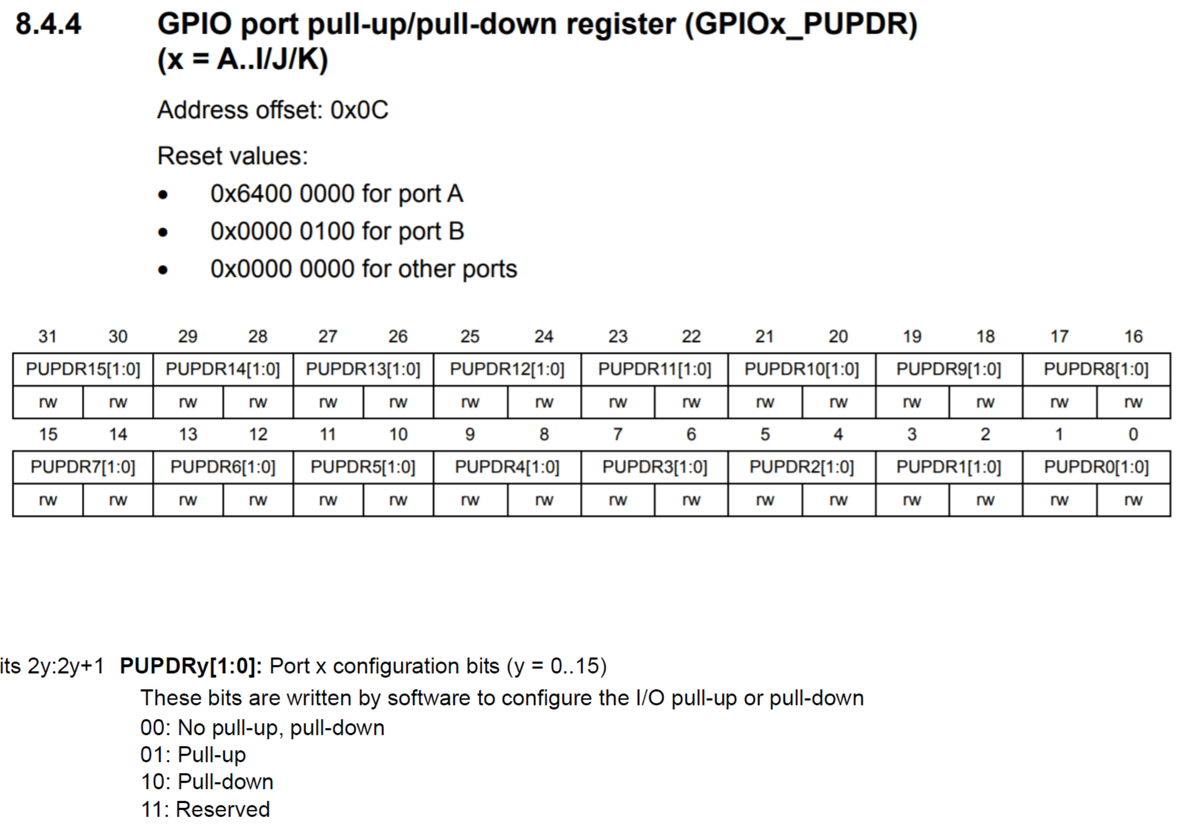
Reset value: 0x0000 0000

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

1.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

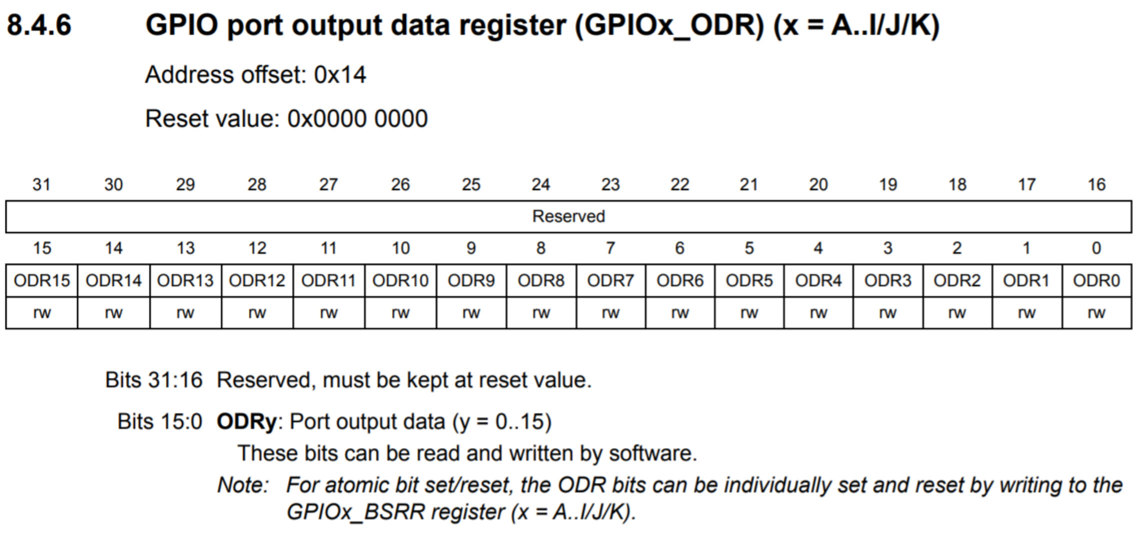
Reset values:

- 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

1.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).

Question 2:

Please evaluate and explain the following statements:

a. An interrupt is always a high priority, urgent task.
b. Using interrupts is always faster than polling.
c. System latency is always larger than interrupt latency
d. Global variables used within an ISR should always be declared volatile.
e. The interrupt vector table must be placed in a specific location in memory.
f. An ISR can return a value and take arguments
g. It is OK for an ISR to safely access the SPI bus that has multiple slaves.
h. If you enable an interrupt in your C code, but don't define the ISR, what code executes when the interrupt is triggered?
i. According to the datasheet of our ARM CORTEX F4, which pins can fire pin change interrupts?

Question 3:

a. Draw the timing diagram for SDA and SCK that shows a master writing two bytes of data to slave at address (7 bit address) 0x13. The first byte is 0x50 and the second byte is 0x33.
b. Show the timing diagram like in a., but for a simple single byte "echo" transaction. That is, the master sends one byte, "A" for example, and the slave sends back the same "A" to the master.

Please be as precise as you can with the timing diagrams!!!!

Question 4:

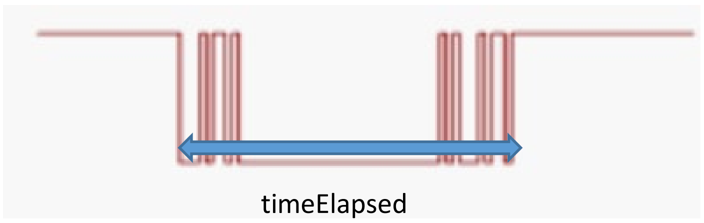
Suppose you are required to retrieve the temperature from the Analog Devices AD5933 chip, which is a chip used to make very precise impedance measurements. The interface is I2C and the relevant portions of the datasheet for this chip are attached. Also assume you have access to several I2C functions as follows:

a. `int Start_I2C()`
   i. Initializes all of the I2C hardware pins etc. Returns 1 if successful, 0 otherwise
b. `int I2C_Write_Byte(uint8_t data)`
   i. Writes 'data' on the I2C Bus. Returns 1 if slave ACKs, or 0 if slave NAKs
c. `int I2C_Send_Start_Condition(uint8_t address, int isRead)`
   i. Send a start or restart condition, followed by 7-bit address, followed by 'isRead' bit, which is 0 if write, 1 if read.
   ii. Returns 1 if successful, 0 otherwise
d. `int I2C_Send_Stop_Condition()`
   i. returns 1 if successful, 0 otherwise
e. `int I2C_Request_Read(unint8_t *buffer, int numBytes)`
   i. reads 'numBytes' bytes off the bus, each followed by a master ACK except for the last byte, which is followed by a master NAK indicating an end to the read. Returns 1 if it gets all the bytes, 0 otherwise and buffer holds the bytes.

Using the datasheet provided, write a C code function `float GetTemperature()` required to get the temperature of the AD5933 chip. You can add any additional variables as long as you state what they are for and be sure to state any assumptions made.

Question 5:

The requirement for this question is to use interrupts to measure the time elapsed while a single button (GPIO) is being pressed. The GPIO pin is pulled to a high state when the button is not depressed. In this application, the button is mechanical and experiences significant bouncing. The goal is to measure the contiguous time (in ms) between when the button is pressed and when the button is released, excluding the bouncing. See diagram below.



You have already setup 3 interrupt handlers. One handler is the ISR that handles Timer0's overflow. That timer is set up with a prescaler of 32, a counter TOP value of 249, and is set up to reset to 0 and count up to TOP. The CLK is running at 8MHz. Here is the handler definition:

```
void  Timer0_OV_Handler()
```

The other 2 handlers are for the pin that is connected to the button. One handles the RISING transition, and the other handles the FALLING transition. Here are the definitions:

```
void pinRising_Handler()
```

```
void pinFalling_Handler()
```

Write the required code snippits (in each handler) that assigns the variable `timeElapsed` to the specified time in the timing diagram. You may add any additional variables, but **be sure to state any assumptions.**