

EECS 373 - Homework #3

Name: unique name:

This assignment is optional and there will be no penalty for not submitting it. If you complete it, the grade will be averaged in with your other homework grades. Due 11:59am on 22 March via Gradescope. We'll release solutions shortly after the deadline to enable review before the second midterm exam. Please use the answer boxes provided for text answers. You can upload a scan or separate page for the diagram. Typed and neat handwritten answers are both acceptable.

Question 1

Short answer questions: **[20 points, 4 each]**

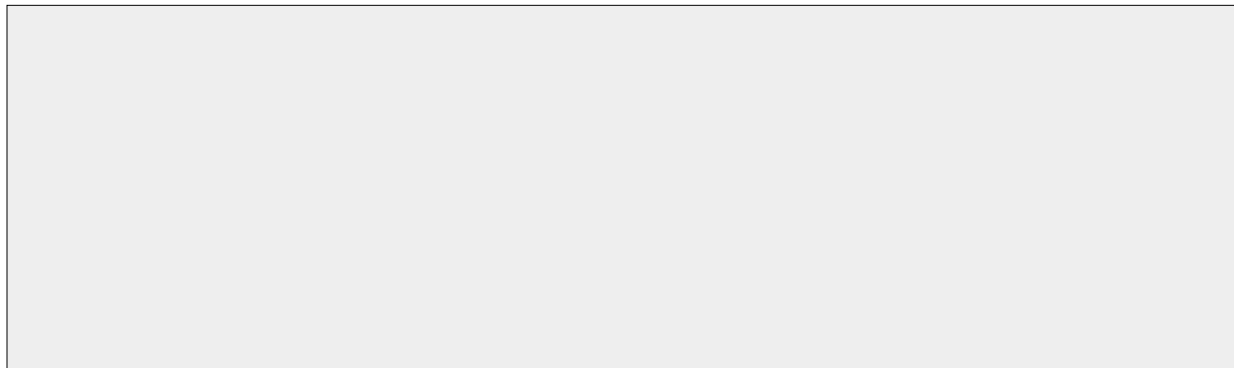
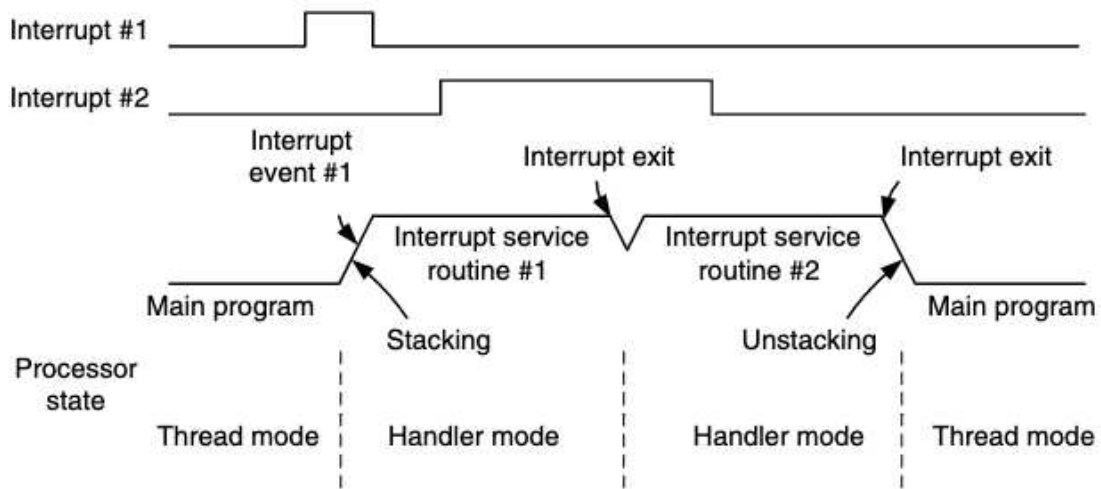
A) What is an interrupt vector table?

B) Why should the time to execute an ISR be as short as possible? How many clock cycles does it typically take to enter an ISR?

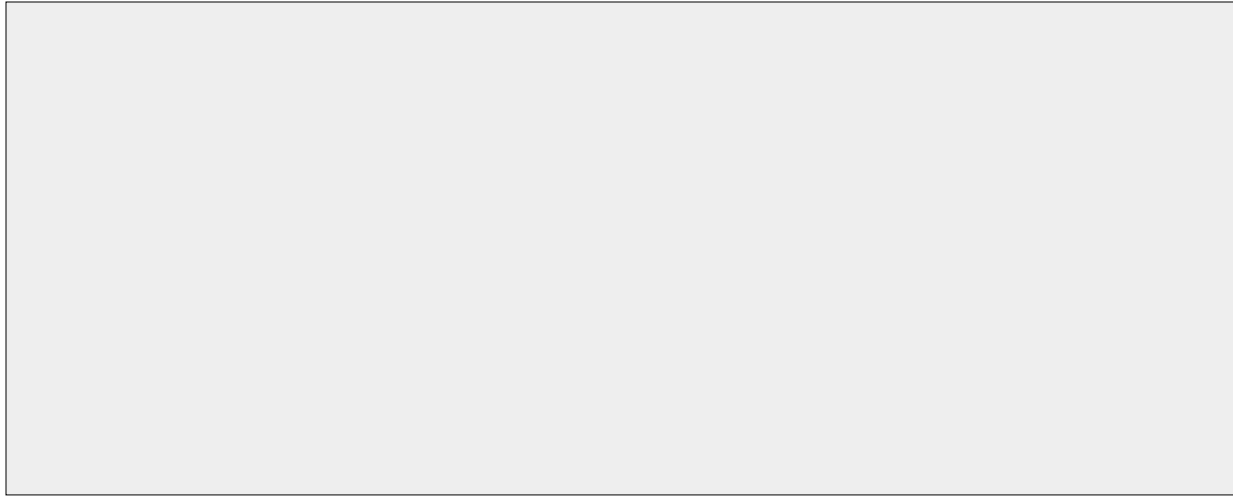
C) What are the hexadecimal addresses of (1) Reset_Handler and (2) USART1_Handler?



D) For the diagram below, which interrupt has the higher priority and which method is being used to improve interrupt latency. Must answer both parts to get full credit.

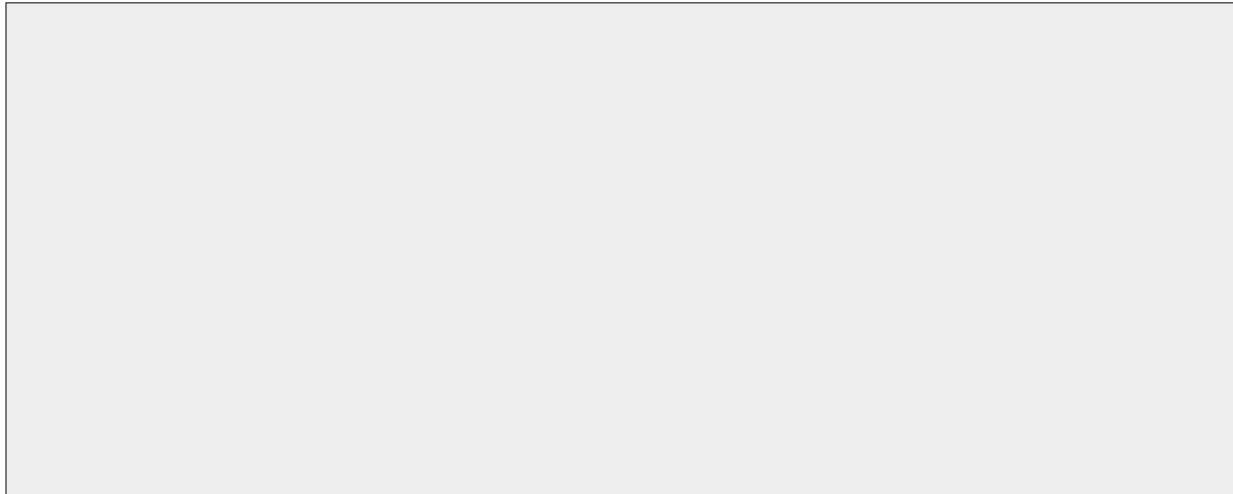


E) When an interrupt signal is generated, what will be done by the processor automatically before the processor starts to execute the corresponding interrupt service routine?



Question 2 [10 points]

A) (5 points) Explain the difference between a *Capture* timer and a *Compare* timer, and give examples of what they are used for.



B) (5 points) Pulse Width Modulation. Suppose a HSE (High Speed External Clock) of 16MHz is selected as the clock of the timer. In order to generate a 1Hz square wave with a duty cycle of 50%, how would you set up the timer? Indicate your counting mode and show the value of ARR, CCR and PSC registers.

Question 3 [10 points]

You are working on a design for our STM32L4R5 which has 5 interrupt sources: A, B, C, D, and E. Recall that the STM32L4R5 only implements the 4 highest priority bits, the other 4 are ignored. You want the following to be true:

- A should be able to preempt any interrupt other than B and C.
- B should be able to preempt any interrupt.
- C should be able to preempt any interrupt other than A and B. C should have a priority higher than A.
- D should be able to preempt only E.
- E should not be able to preempt anything.

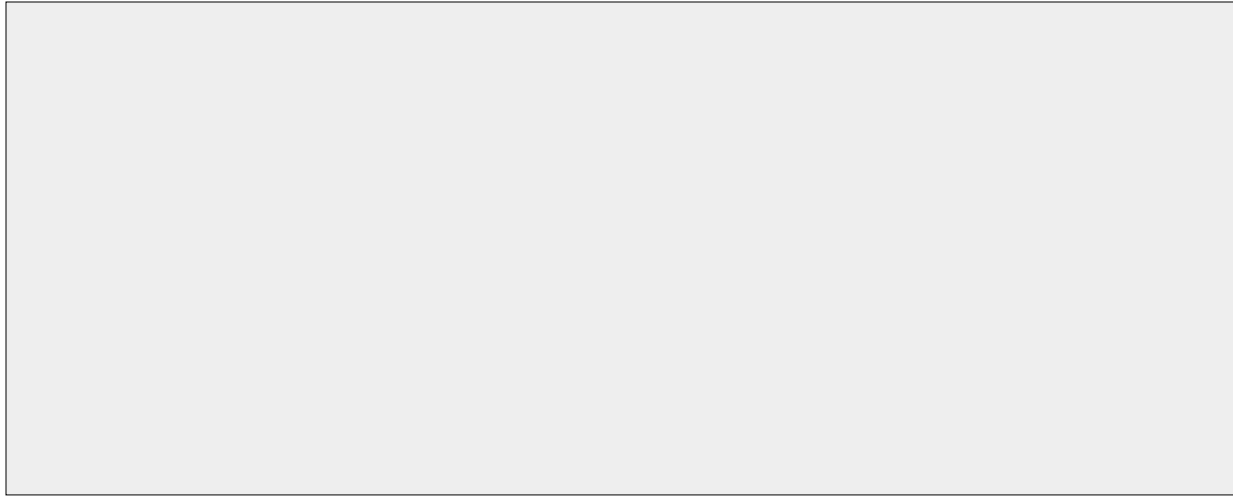
Table 51. Priority grouping

PRIGROUP [2:0]	Interrupt priority level value, PRI_N[7:4]			Number of	
	Binary point ⁽¹⁾	Group priority bits	Subpriority bits	Group priorities	Sub priorities
0b0xx	0bxxxx	[7:4]	None	16	None
0b100	0bxxx.y	[7:5]	[4]	8	2
0b101	0bxx.yy	[7:6]	[5:4]	4	4
0b110	0bx.yyy	[7]	[6:4]	2	8
0b111	0b.yyyy	None	[7:4]	None	16

1. PRI_n[7:4] field showing the binary point. x denotes a group priority field bit, and y denotes a subpriority field bit.

Part A [4 points]

List **all** PRIGROUP setting or settings you could use in this case. Assume no two interrupts can be assigned the same priority. **Provide your answer in 3-digit binary and explain. [3]**



Part B [6 points, one each]

Assuming you are using the STM32L4, indicate, ***in 4-bit binary***, what priorities you will assign to each interrupt. Let us know which PRIGROUP setting you are using (mainly if you have more than one PRIGROUP listed above). Again, no two interrupts may be assigned the same priority.

PRIGROUP= (3-digit binary)

A priority= (4-digit binary)

B priority= (4-digit binary)

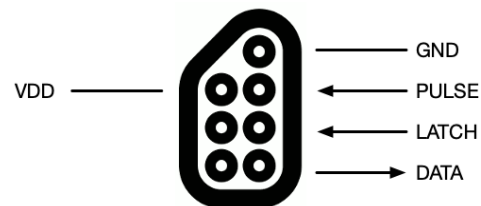
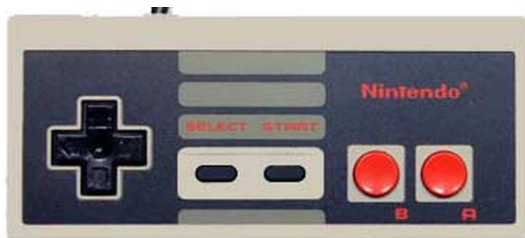
C priority= (4-digit binary)

D priority= (4-digit binary)

E priority= (4-digit binary)

Question 4: NES Controller Design

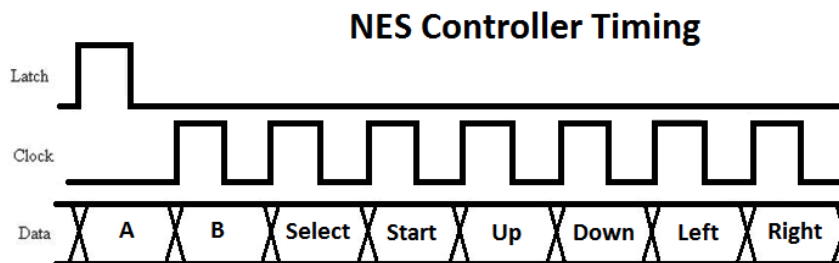
The NES (Nintendo Entertainment System) game controller is essentially a shift register that loads the state of the 8 buttons on the rising edge of the “LATCH” signal and shifts the value out serially on the “DATA” line with subsequent rising edges of “PULSE” as shown in the timing diagram below. The data is easily read by the game processor with another shift register. The data is persistent at each transition so each button value can be latched and shifted into a shift register on each falling edge of pulse. However the first value, must be latched in using the *Latch* signal instead of the Pulse signal. For example, the **A** button can be read on the falling edge of *Latch*, the **B** button can be read on the first falling edge of Pulse, the **Select** button read on the second falling edge of pulse and so on.



Pin Description

Pin #	Pin Name	Pin Type	Function
1	GND	Power	Ground
2	PULSE	Input	Pulse (aka Clock)
3	LATCH	Input	Rising edge causes state of the buttons to be latched into the NES game controller
4	DATA	Output	
5	NC	-	No Connect (pin not used)
6	NC	-	No Connect (pin not used)
7	+5V	Power	Power

Timing Diagram



Your task is to design a system that will read the NES game controller using interrupts. You should read the entire problem thoroughly before starting.

Your task is to design a memory-mapped IO interface to the N8 that will capture the button values and generate a hardware interrupt signal “NES-INT” after each read is done.

- Writing to location 0x40050000 is to cause the start of a read of the NES controller. Once the read has been done an interrupt should be generated letting the core know that the value is ready to be read. The data sent with the write to this address does not matter. You may assume no read to this location will occur while in the process of doing another read.
- Reading from location 0x40050000 should return the value of the last complete transaction of the controller. This should be a 32-bit binary value where the lower 8 bits indicate the value of the buttons (A being the msb and Right being the lsb of that 8-bit value). You may assume no write will occur to this location while a read to that location is in progress.

Once that hardware has been designed, you are to write two C functions.

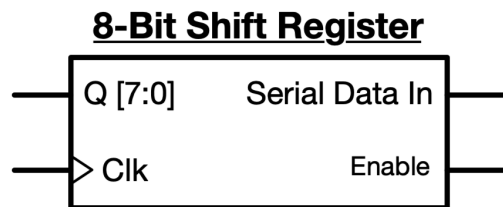
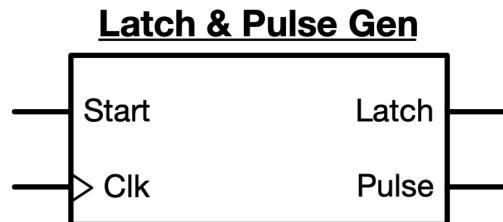
- An ISR that gets called when “NES-INT” is asserted and puts the 8-bit value read from the NES controller into a global variable called “NES_value” and sets the global variable “NES_ready” to 1.
- A C function that when called checks the value of NES_ready. If that value is a 0, it is to just return a -1. If instead NES_ready is a 1 it is to:
 - o Set NES_ready to be a 0
 - o Start a new read of the NES controller.

Part 1: NES Module [25]

Note: The logic which generates the Latch and Pulse signals is given to you (**you don't have to design it!**).

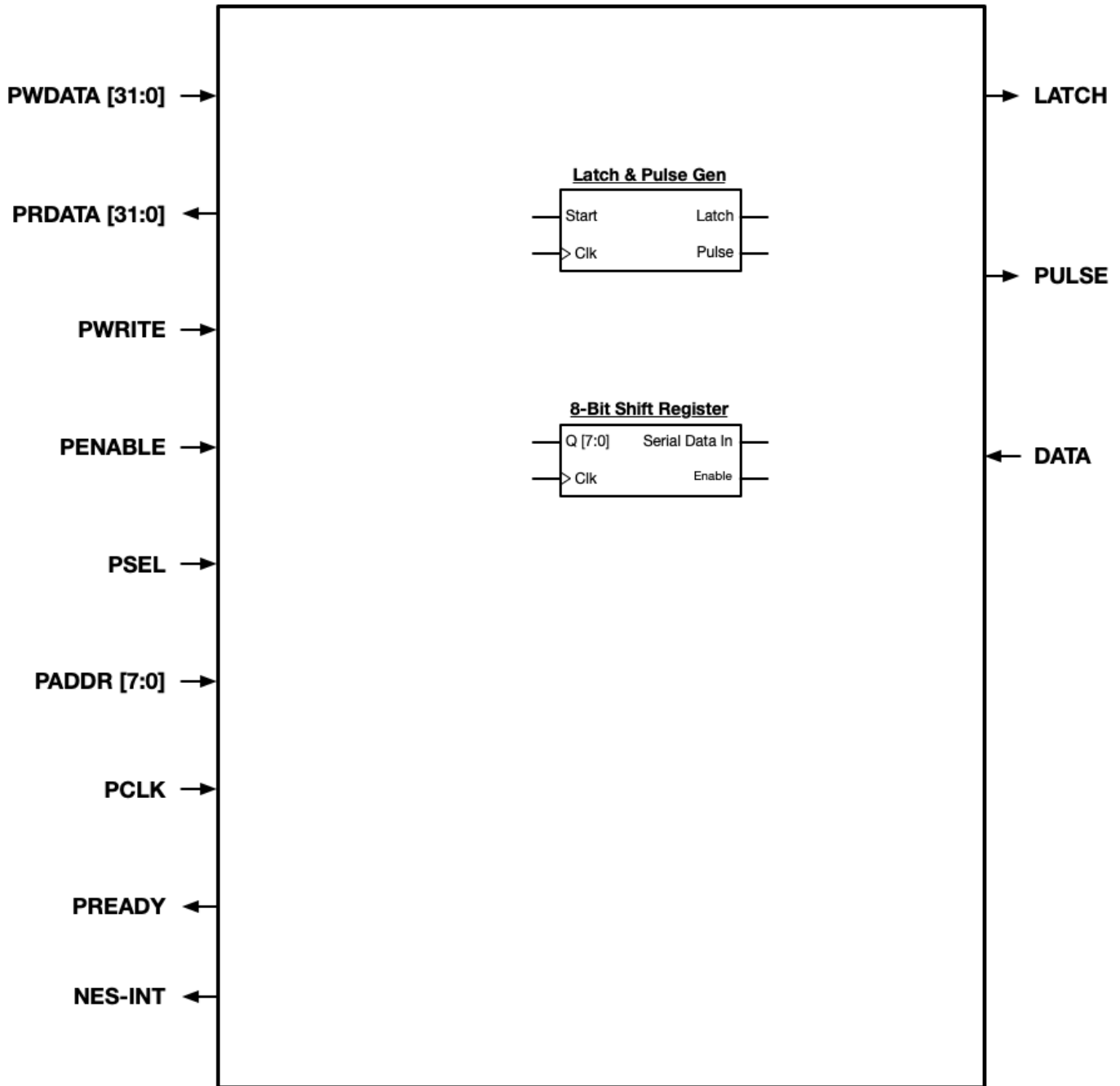
You are to design this module in schematic form. The following components are available. You may use as many of each device as you need unless otherwise indicated.

1. One serial shift register with a serial data input, an enable, and a clock input. It has an 8-bit parallel output that reflects the values in the shift register. The register shifts data to the right on every rising edge in which EN is 1.
2. One Latch/Pulse device. It generates Latch and Pulse as described whenever the signal "Start" is found to be asserted ("1") on the rising edge of it's clock. It will start immediately. PCLK must be given as its clock input. This device will not function correctly if "Start" is asserted while the device is in the process of generating latch and pulse.
3. AND, OR, XOR and NOT gates.
4. Tri-State drivers.
5. DFFs with Enable (clock enable) and DFF connected as a registers
6. You may also you a "AND Mask" to for the PAddress to simply drawing complicated AND gates.



You may represent buses as a single wire, but indicate which signals it carries. For example, PADDR[3:0]. Further, you may show signal connections with a signal label instead of a line. For example, you can write PCLK wherever a PCLK is needed rather than drawing lines connecting all the places PCLK is needed.

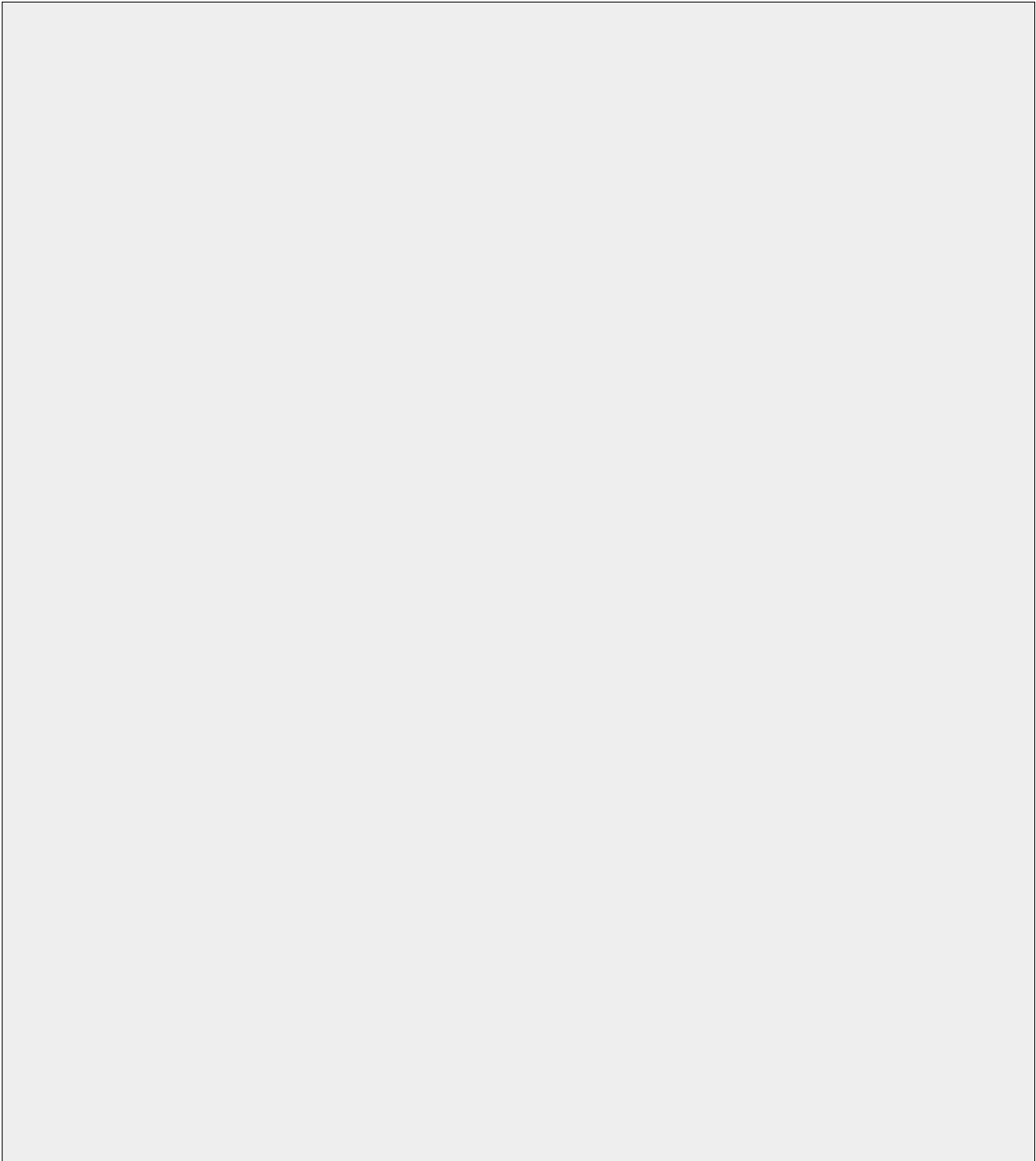
Use the space on the following page to draw your schematic. All inputs are shown on the left and all outputs on the right.



Part 2: Interrupt Handler [10 points]

Write an ISR that gets called when NES-INT is asserted and puts the 8-bit value read from the NES controller into a global unsigned char called "NES_value" and sets the global int "NES_ready" to 1.

```
__attribute__((interrupt)) void IRQHandler(void) {
```



Part 3: Read function [10 points]

Write a C function called "NES_read" that when called checks the value of NES_ready. If that value is a 0, it is to just return a -1. If instead NES_ready is a 1 it is to do the following tasks:

- o Set NES_ready to be a 0
- o Start a new read of the NES controller.
- o Return the value last read from the NES controller.

```
int NES_read(void) {
```

