

Homework 4

Assigned: Thursday 1/28; Due: Thursday 2/4 at 3:10pm

The purpose of this assignment is to:

- Review interrupts and priority/preemption issues on our SmartFusion in lab. (22 points)
- Act as a refresher on how to work with pointers, especially function pointers, in C. (20 points)
- Touch on the new concept of weak references (first seen in lab 4) (8 points)

1. Look at http://www.eecs.umich.edu/courses/eecs373/readings/Actel_SmartFusion_MSS_UserGuide.pdf starting on page 10.

a. How many different external interrupts does the NVIC on the SmartFusion support? [2]

b. What are the external interrupt numbers for FABINT, GPIO_0, and GPIO_1? [3]

c. If you want to *enable* the FAB_INT and TIMER_1_IRQ, what value(s) will you write to what memory location(s)? [3]

2. Look at page B1-18 of http://www.eecs.umich.edu/courses/eecs373/readings/ARMv7-M_ARM.pdf. In your own words, explain what priority grouping is. Your answer must include the word “preemption”. [5]

3. You are working on a design for our SmartFusion which has 5 interrupt sources: A, B, C, D, and E. Recall that the SmartFusion only implements the 5 highest priority bits, the other 3 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 and should have a higher priority than E or F.
- F should be able to preempt only E.

PRIGROUP[2:0]	Binary point position	Pre-emption field	Subpriority field	Number of pre-emption priorities	Number of subpriorities
b000	bxxxxxxx.y	[7:1]	[0]	128	2
b001	bxxxxxx.yy	[7:2]	[1:0]	64	4
b010	bxxxxx.yyy	[7:3]	[2:0]	32	8
b011	bxxxx.yyyy	[7:4]	[3:0]	16	16
b100	bxxx.yyyyy	[7:5]	[4:0]	8	32
b101	bxx.yyyyyy	[7:6]	[5:0]	4	64
b110	bx.yyyyyyy	[7]	[6:0]	2	128
b111	b.yyyyyyyy	None	[7:0]	0	256

- a. 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]**

- b. Indicate, **in 8-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. [6]**

- PRIGROUP=_____ (3-digit binary)
- A priority=_____ (8-digit binary)
- B priority=_____ (8-digit binary)
- C priority=_____ (8-digit binary)
- D priority=_____ (8-digit binary)
- E priority=_____ (8-digit binary)
- F priority=_____ (8-digit binary)

For problems 4 and 5 you will be again writing C code using pointers and handing it in via a link provided on a Google sheet. Problems 1, 2, 3 and 6 will be turned in via Gradescope. Problem 6 asks questions about the code you will work with for problems 4 and 5. Problems 4 and 5 will be graded together and are worth 20 points as follows:

- 20 points: Everything works.
- 10 points: You put effort in but it doesn't quite work.
- 0 points: Little to no effort / nothing works / nothing submitted.

Honor Code Reminder: In the interest of making this easy for you to implement, test, and submit this is probably one of the easiest assignments to cheat on ever. Please don't. Resist the temptation to "glance" at another solution for help because you got stuck. There just isn't enough code that a "glance" will do anything but give you the solution. This assignment is deliberately worth very few points; if you didn't do it, just don't submit anything [why risk cheating?]. The point of this assignment is to make your life easier later this semester, don't cheat yourself.

Grab a copy of the code from <https://github.com/eecs-373-w16/HW4>. We recommend forking this repository to do your work.

First things first, type `make main && ./main` and look at the output. Look over `main.c` and `sort.c` to try to understand what the current code is doing. Specifically,

- How does main call the different sort functions in `sort.c`?
- How does main know how many sort functions there are? What is the purpose of the compare function?

4.

Add a third sort algorithm. Instead of implementing it yourself, use the built-in `qsort` from the C standard library. For details on `qsort`, type `man qsort`. The `qsort` type signature doesn't quite match the `sorting_fn` type signature, so you will need to write a wrapper function.

Do your work for problem 4 in the `sort.c` file. You may only edit `sort.c` for this problem. When you are done, type `make check_main` to check your work.

5.

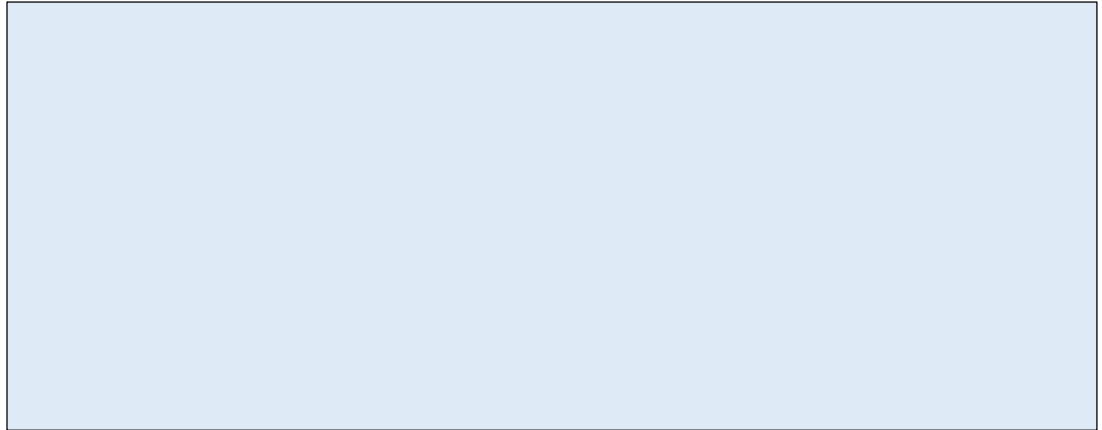
Next you will modify the sort functions to reverse the order they are sorting in. You will do this without modifying `sort.c`. Type `make reverse && ./reverse`. Currently this will behave the same as main. The difference between main and reverse is that reverse also links in `reverse_sort.c`. Add a function to `reverse_sort.c` so that numbers are now sorted in descending order. You may find it useful to consult Lab 4 for a refresher on weak links (also called weak references or weak symbols) and/or you may wish to read https://en.wikipedia.org/wiki/Weak_symbol.

Do all of your work for problem 5 in `reverse_sort.c`. When you are done with this problem, type `make check` to check your work.

Notes

As the readme file says, this has only been tested on the CAEN Linux load. It may work on other x86 Linux machines (and in fact probably will) but we won't support it on any other machine. See the readme file for details about how to turn this in.

6. These questions refer to the code from problems 4 and 5
- a. The code measures the "effort" of each sorting algorithm. What is effort in this case (what is it actually measuring)? **[4]**



- b. The end of main is printing something out. What values is it printing out and what are those bytes? **[4]**

What it is printing:



What are those bytes? Explain clearly.

