CS 431-1 Embedded Systems

Dashboard / My courses / CS 431-1 / 15 November - 21 November / HW 4

HW 4

Opened: Wednesday, 29 September 2021, 12:00 AM
Due: Monday, 29 November 2021, 7:59 PM
To do: Make a submission

I HIGHLY ADVISE YOU TO WORK IN AN EDITOR AND THEN COPY PASTE YOUR CODE INTO THE BROWSER!
You can imagine what will happen to your code if you make the browser crash. This is also a good time for browser extension advertisement.

Working Draft! - Keep checking here for changes.

This hw uses https://simulator.mbed.com. Get yourselves acquainted with the simulator platform by reading the docs at https://mbed.com and playing around with the examples already there. But, better yet; dont just lean on an online service for your homeworks. Chances are it will be turned off, it'll crash and whatnot. My genuine advice is that you should spin up your own Arch based linux VM (my preference is manjaro, and follow the setup script at [(v4) Virtual Machine w/ Keil uVision, edsim51 & mbed-simulator (v4)]. Or download the [Virtual Machine w/ Keil uVision, edsim51 & mbed-simulator] that I shared but make sure to (git) pull the *mbed-simulator* repo to its latest commit.

- Mbed simulator introduction. **READ THE LIMITATIONS!** https://os.mbed.com/blog/entry/introducing-mbed-simulator/
- mBed API https://os.mbed.com/docs/mbed-os/v5.15/apis/index.html
- **InterruptIn** https://os.mbed.com/docs/mbed-os/v5.15/apis/interruptin.html
- **PwmOut** https://os.mbed.com/docs/mbed-os/v5.15/apis/pwmout.html
- **Ticker** https://os.mbed.com/docs/mbed-os/v5.15/apis/ticker.html
- **Timeout** https://os.mbed.com/docs/mbed-os/v5.15/apis/timeout.html

Your next hw assignment is to build a device that changes the brightness of an external LED component with respect to the number of button presses it receives within the last 5 seconds. 1 press will turn it off, and 10 will set it to max. So you should set brightness to zero when only 1 clap is heard and then 2 claps is minimum brightness, and then it goes linear (or whatever you want) from there to 10.

There are 2 ways to check how many button presses you received in the last X seconds. First method involves creating a clock and recording the timestamps of the last 10 button presses, and checking how many of the last button presses are within X.

You should periodically check if you received an external interrupt within the last X milliseconds. This is because the output of the digital module is -yes- discrete in amplitude(1,0) but not discrete in time. It's continuous time. And then you should buffer if an intr. was received or not.

LEDs must preserve their brightness values, they should NOT just automatically turn off just because there were no claps. The implication here is that you should turn them off only if the clap count you keep reaches 1 from 0 not from 2. That is what we're gonna implement in this hw. *In case you're curious about the second method; scroll down below to the "if you're feeling adventurous" part.*

Demonstrate that;

1. You can set the brightness of an LED (you'll need an external one, see demos) with a predefined (const, precompiler def., etc.) value and that you can also turn it on/off.
2. ~~Use the time API mbed provides~~ Time api seems to be broken. Implement your own timekeeping utilities via Ticker/Timeout etc. and print the time everytime button is pressed. Remember NOT to use printfs in an ISR. But DO capture the time within the ISR. (there you go your first shared-data).
3. Create a data structure that can hold the last 10 presses and initialise it in such a way that the system starts with 0 brightness. You may wanna use a typedef struct, or a class if you're feeling adventurous because further down the line we'll increase the number of LEDs. Let's call them clockbuffers.
4. Study the "button bouncing" problem. We're gonna implement a safeguard for that; Demonstrate that every time button is pressed, IRQ for that specific button is disabled for BUTTON_DEBOUNCE_TIME_MS amount of time. Demonstrate this functionality by using exaggerated debounce times like 500ms. It really should be around 20ms.
5. Create a function that takes in a pointer for a clockbuffer, and then returns a brightness value between 0.0f and 1.0f. Demonstrate that you can use this function and the demo of step1 to set brightness to your LED. Do NOT pass the entire struct/class as a parameter!
6. Now create a *task* so that func in 5 is run when button is pressed. I.e. this *task* will replace/append to step 2.
7. Hopefully you've followed at least a very simple OOP approach, because now you should demonstrate you can do this for multiple buttons. Create an array of structs/classes that can hold button-LED pin pairs along with some clockbuffers. This array's size should be fixed (precompiler macro) and button-LED pin pairs should be initialised from precompiler macros. time-buffers should be initiliased as before.

And of course; now your task in step6 should check all buttons. Or better yet; you should now have more than a single task that checks the button flags. *Let's see if anyone's going to create an array of functions; it's cool but not necessary. A for loop calling the same function with different parameters is pretty much the same.*

8. Final task: "*preparing for power-failures*"; even the simulator provides some sort of persistent memory (see the block device) example. Every time the configuration changes write the config onto block device so that it persists between browser refreshes (i.e. power failures). Obviously you should also read from block-device when the device is initialised. The catch here is -since you'll practically copy-paste code from block-device example- you should implement an error checking algorithm to validate the memory contents in block device. And if it's corrupt you should re-init everything as you did in step7. Demonstrate this functionality by writing weird stuff onto BD and observing your device initialises "dark".
9. Final final task; If you haven't read the limitations page, enjoy having to restart your browser. A bug of the simulator is a feature for me to enforce you to put your CPU to sleep when it's not doing anything. Make sure that your main while loop sleeps a little bit. Be ready to explain what happens if you do overdo it (think of a task that takes too long; one of the shortcomings of RRw/ISR that we mentioned).
10. Unrelated to above: play around with scanf, getchar and similar to observe the behaviour of serial data capture in this simulator (it's awful). And be ready to answer what you observed.
11. **I DONT WANT TO SEE ANY UNPROTECTED DATA SHARING! See CriticalSectionLock - APIs | Mbed OS 5 Documentation**

If necessary, submit different C files for each part or be ready to direct your TA to make necessary modifications to demonstrate each part. I suggest uploading different files to be honest, make it easy for everyone. Each part is 10 points, and demo questions are 20 (step9 - step11 dont count).

REMARKS: Unlike the previous hw; programming busy-waits are NOT allowed. Please use Timers/Timeouts/Tickers or any other method of non-busy waiting. Also note that you are only allowed to use official mbed libraries and NOT the user/community libs/codes. You are also asked to use external interrupts (InterruptIn) to count external events rather than polling. **Also it turns out mBed's circular buffer has nothing to do with a circular buffer and it's a simple stack. Do not try to use it in my opinion. Thx to one of your colleagues.**

Do note that this also implies you should use a **RR w/ISR** architecture. And for your own sake, design first code second.

If you're feeling adventurous this is the last year's hw reaching for the same goal:

1. You can set the brightness of an LED (you'll need an external one, see demos) with a predefined (const, precompiler def., etc.) value and that you can also turn it on/off.
2. You can use a timer by periodically printing a string which indicates if clap-button was pressed within last X seconds.
3. You can buffer the clapped or not-clapped values into a Ring-Buffer (circular buffer) of prechosen size, by printing this buffer out in binary or (... for 0 and ''' for 1) values when another button is pressed (you can add as many components).
4. You can count the number of external interrupts from clap-button within the last X seconds. This count will count the clap values in the last k*X seconds, but then it'll also start to decrease as time goes by silently. Because this will be a running value calculated periodically based on the last k*X seconds of any given time.
5. You can then hold on to the maximum count as this will be a representative of our brightness value. So, demonstrate that you can hold on to this value by; keeping another value that increments as your count goes up but does not go down with it. And you also only reset this value when your count goes up to 1 (0 brightness, LED off). This will be the time to set our brightness values reflected on that max.
6. You can set the brightness of LED with respect to the count value in 4.
7. You can set the brightness of LED with respect to the better value in 5.
8. Unrelated to above: play around with scanf, getchar and similar to observe the behaviour of serial data capture in this simulator (it's awful). And be ready to answer what you observed.

If you feeling extra adventurous after all this work feel free to take a look at "Events" demo which uses a function-queue-scheduling architecture and adopt that. FYI next hw will already involve using a function-queue-scheduler.

If you got any questions, ask them in HW 4 Discussion

Submission status

Submission status	No attempt
Grading status	Not graded
Time remaining	2 days 23 hours
Last modified	-

Submission comments Comments (0)

Add submission

You have not made a submission yet.

◄ HW 3 Discussion

Jump to...

HW 4 Discussion ►