Lab 1 - Memory-Mapped IO and Interrupts

Goals

- Create a GPIO driver using memory-mapped I/O
- Explore interrupts

Equipment

- Computer with build environment
- Micro:bit and USB cable

Documentation

- nRF52833 datasheet: https://infocenter.nordicsemi.com/pdf/nRF52833_PS_v1.3.pdf
- Microbit schematic: https://github.com/microbit-foundation/microbit-v2-hardware/blob/main/V2/MicroBit_V2.0.0_S_schematic.PDF
- Lecture slides are posted to the Canvas homepage

Lab 1 Checkoffs

You must be checked off by course staff to receive credit for this lab. This can be the instructor, TA, or PM during a Friday lab session or during office hours.

- **Part 1:** Setup
    a. Demonstrate the error app running and the message it prints

- **Part 2**: Using Memory-Mapped IO to control GPIO
    a. Demonstrate code that controls the Microphone LED with raw MMIO addresses
    b. Show your MMIO struct and library code in gpio.c
    c. Show your application code in main.c
    d. Demonstrate your application controlling the LED with buttons

- **Part 3**: Interrupts
    a. Demonstrate triggering an interrupt with GPIO
    b. Show your application code in main.c
    c. Demonstrate your application showing preemption of interrupts

Also, don't forget to answer the lab questions assignment on Canvas.

Lab Steps

Part 1: Setup

1. Find a partner

- Rule: you can pick any partner you want, but you can't pick the same partner twice
- You MUST work with a partner
    - We don't have enough computers otherwise

2. Set up your computer

- Log into Windows
    - Password: 327-19s
- Open VMWare Workstation Player
- Open the virtual machine: CE346
    - It'll load for a minute and then ask you to log in
    - Password: microbit

3. Create your Github assignment repo

- Follow this link: https://classroom.github.com/a/otMQx_PJ
- Pick a team name
- Pick your partner
- Generally, do what it says
- At the end, it should create a new private repo that you have access to for your code
    - Be sure to commit your code to this repo often during class!
    - If your computer crashes, all your files WILL BE LOST unless committed and pushed to Github.
- That link will 404. You first have to go to https://github.com/nu-ce346-student and join the organization
- Also create a personal access token:
    - Go to your github profile -> Settings
    - Then Developer Settings on the left
    - Then Personal Access Tokens on the left
    - The click the Generate New Token button on the top right
    - Add a note that is the name of this token (not important, type anything)
    - IMPORTANT: check the repo checkbox below the name of the token
    - Then scroll to the bottom of the screen and click the Generate Token button
    - This will create a password that allows you to clone repos
        - It will only show this once, so take a picture on your phone or something
            - Copy-pasting it into a google doc in your personal drive would probably be useful

        - It will be in gray at the top of the screen

4. Clone your lab repo locally

- Open a terminal
- You can clone the repo right to the home directory of the computer
    - Remember, everything in this VM will disappear when it powers down
- At the top right of your shiny new private repo, there is a green button that says code. Copy the HTTPS link to your git repo from there.
- `git clone <YOUR-REPO-HTTPS-LINK-HERE> --recursive --shallow-submodules`
    - Remember to include both of those flags!
    - Recursive is necessary to clone submodules
    - Shallow submodules makes it like five minutes faster to run

5. Program a board

- Plug the board into the computer
    - WARNING: if you haven't loaded code on it before, the default app makes noise
        - And is rather annoying
    - You plug into the USB on the top of the board
- Attach the board to the VM
    - A pop-up might appear asking you where to attach the device. Attach it to the VM. If not:
        - In the menubar, click Player/Removable Devices/Segger-JLink (out of your USB devices)
        - If you hover over Player/Removable Devices/ again, it should be checked
        - You'll have to check this button each time you plug in a board. There will be a separate one for each board you have attached to the computer.
- In the blink app
    - `make flash`
    - It should pop up a window with a loading bar that uploads the code
    - Things like "Downloading file [_build/blink_sdk16_blank.hex]..." and "O.K." are good
    - Things like "J-Link connection not established yet but required for command" and "Connecting to J-Link via USB...FAILED: Failed to open DLL" are bad
    - Also, the board should start blinking the red microphone LED if it works

6. Get some apps working

- There are three good starter apps:
    - blink - blinks the microphone LED
    - printf - periodically prints a message from the board
    - error - demonstrates a hardfault and error messages on the board

- Commands to control them
    - make flash
        - To build code and load it onto the board over JTAG
    - miniterm /dev/ttyACM0 38400
        - To listen to serial output
        - (Any other serial console would work too)
        - Note: it doesn’t buffer output. Anything that happened before you opened it won’t appear. Hit the “Reset” button at the top of the Microbit to start the currently loaded program again.
        - Also note: you don’t have to close this when programming a board. Just leave it open in another terminal window. It should only stop working if you unplug your Microbit.
- Take a look at each of the starter apps and try out modifying board behavior

- **CHECKOFF:** demonstrate the error app running and the message it prints

Part 2: Using Memory-Mapped IO to control GPIO

7. Use raw pointers to control an LED

- Look through the section on GPIO in the nRF52833 manual. It starts on Page 138
    - Particularly take a look at the registers for the GPIO peripheral
- Start with the application at software/apps/gpio/
- Enable the Microphone LED with raw memory-mapped IO addresses
    - The Microphone LED is Port 0, Pin 20 and is active high
    - You will need to write to the DIR and OUT registers (in that order)
        - Alternatively, the SET/CLR versions of those
    - To write an individual bit, you'll need the bit shift operator **<<** https://www.arduino.cc/en/pmwiki.php?n=Reference/Bitshift
    - This should only take two lines of code
    - Take a look at the `apps/temp_mmio/` example app for syntax
- **CHECKOFF:** demonstrate this code to course staff

8. Implement GPIO library

- Code for the GPIO driver library goes in gpio.c and gpio.h.
- First, create a struct GPIO MMIO registers
    - The GPIO register definitions can be found in the GPIO section of the nRF52833 datasheet, which starts on Page 138.
    - Each type should be a `uint32_t`
    - You can use arrays of `uint32_t` to specify gaps in the address space
    - You can also use arrays of `uint32_t` to specify repeated registers (such as PIN_CNF)
    - Be sure to use the `volatile` keyword when actually instantiating your structure pointer as a global variable.
    - You'll need two struct pointers, one for each port
        - Alternatively, an array of two struct pointers
- To test that your GPIO MMIO register struct is correct, print out the address of a few registers and double-check against the datasheet
    - You will have to print them inside of a function in gpio.c
    - You can print pointers with the format specifier %p
    - The following code takes the address of a struct member: `&(struct->member)`
- Implement the functions in gpio.c using your MMIO struct.
    - Configuring a pin as an input requires both setting its direction and connecting the input buffer. Both can be done with the appropriate PIN_CNF register
    - Each GPIO pin number is a combination of Port `(0 or 1) << 5` and pin number (0 to 31)
        - You'll need to determine which struct pointer to use based on the port
    - To set individual pins, you'll need to use bit masks using a combination of the **&**, **|**, and **~** operators https://www.arduino.cc/en/Tutorial/Foundations/BitMask
- **No checkoff**: continue to the next step

9. Control LED with buttons

- Use Button A and Button B to control the Microphone LED. One should turn the LED on and the other should turn the LED off
    - Use your GPIO library to read the buttons and control the LED
    - Button A is P0.14 and is active low
    - Button B is P0.23 and is active low
    - If code isn't working, it's time to debug your GPIO library
        - Are the MMIO registers mapped to addresses correctly?
        - Are there additional fields that you do need to write to?
        - Are there additional fields that you shouldn't be writing to but are?
- **Checkoff**: demonstrate your working application to the course staff
    - Also show your code in main.c and gpio.c

Part 3: Interrupts

10. Trigger an interrupt with GPIOTE

- Configure the input pin with GPIOTE
  - The GPIOTE register definitions can be found in the GPIOTE section of the nRF52833 manual, which starts on Page 146.
  - The MMIO struct is already made for you. Access it as `NRF_GPIOTE->REGISTER`
    - For example: `NRF_GPIOTE->INTENSET` or `NRF_GPIOTE->CONFIG[0]`
  - You can use Button A or B to trigger the interrupt
    - Button A is P0.14 and is active low
    - Button B is P0.23 and is active low
  - In the CONFIG register, OUTINIT isn't important since you should be in Event mode
  - Make sure you are setting the `INTENSET` register correctly. Interrupts must be enabled both in the GPIOTE peripheral and also in the NVIC (as explained next)
- Enable the interrupt in the NVIC and set its priority
  - Functions for interacting with the NVIC:
    - `void NVIC_EnableIRQ(uint8_t interrupt_number);`
    - `void NVIC_DisableIRQ(uint8_t interrupt_number);`
    - `void NVIC_SetPriority(uint8_t interrupt_number, uint8_t priority);`
  - Interrupt numbers are defined for you in headers and you can use the names in your code. Relevant numbers:
    - `GPIOTE_IRQn`
    - `SWI1_EGU1_IRQn`
    - `For example: NVIC_EnableIRQ(GPIOTE_IRQn)`
  - Priority is a number from 0 to 7 where a lower number is higher priority (pick anything for now)
- Do something in the handler to show that you're there
  - For this step, the `GPIOTE_IRQHandler()` will be what runs
  - I recommend `printf()`. Loops and `nrf_delay_ms()` can also be used
- Trigger a GPIO interrupt
  - Upload the code that you've written to the board
  - If everything is configured correctly, pressing the Button should trigger an interrupt and cause the code in the `GPIOTE_IRQHandler()` to run
- **Checkoff**: demonstrate that you can trigger an interrupt with GPIO

11. Trigger a software interrupt

- Use the functions `software_interrupt_init()` and `software_interrupt_trigger()` to do this
  - They trigger interrupts through the Event Generation Unit (EGU) peripheral
- You will also need to set the priority of the software interrupt as previously done for GPIO
- **No checkoff**: continue to the next step

12. Nested interrupts

- Make the GPIO interrupt preempt the software interrupt
  - Lower priority numbers take precedence over higher priority numbers
  - Use some combination of a for loop, `printf()`, and `nrf_delay_ms()` to make the software interrupt handler run for long enough that you can press a button and observe the effect
- **Checkoff**: demonstrate preemption occurring to the course staff
  - Also show your code in main.c