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 Steps

1. Update your local repository

- `cd` into the base of your repo
- `git pull https://github.com/nu-ce346/nu-microbit-base.git`
- `git submodule update --init --recursive`
    - There may be no changes

Part 1: Using Memory-Mapped IO to control GPIO

2. 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:** leave this code commented out in your main function to show to course staff

3. 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 manual, 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.
    - Setting a pin as an input requires both setting its direction and connecting the input buffer
    - 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

4. 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 2: Interrupts

5. 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
- 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 (pick anything)
- Do something in the handler to show that you're there
    - I recommend `printf()`. Loops and `nrf_delay_ms()` can also be used
- **No checkoff**: continue to the next step

6. 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

7. 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

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**: Using Memory-Mapped IO to control GPIO
    a. Show your commented out 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

- **Part 2**: Interrupts
    a. Show your application code in main.c
    b. Demonstrate your application

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