CMPE1250
Winter 2025
v1.1

ICA 3
Bitwise Operations

1 Introduction

Bitwise operations are uncommon in your PC programming courses, but they are a fundamental skill in embedded programming. In the embedded world, computing resources are very valuable, so you often use a single bit to store a binary value rather than a whole byte. Your code is also interacting directly with hardware, so you will often need to read or manipulate a single bit within a register. You will also need to shift data left and right for a variety of reasons. In embedded programming, we operate on the level of bits and registers. We do this with bitwise operations, which you will explore in this assignment.

2 Outcomes

By the end of this assignment, you should be able to:

- Create a new Code Warrior project
- Clear, set, and toggle specific bits with bitwise operations
- Explain the difference between bitwise and logical operators
- Shift values within a register
- Mask bits to check the value of individual register bits
- Manipulate single bits in `PT1AD1` with bitwise operations
- Set a breakpoint in CodeWarrior

3 Assignment

In this assignment you will have to perform several very common bitwise operations. To avoid making this a strictly theoretical exercise, you will be writing some code for your microcontroller board that will turn the LEDs on and off. However, you likely haven't learned about the General-Purpose Input-Outputs (GPIO) that are used to make this happen. For that reason, some code has been provided for you. For now, focus on understanding the bitwise operations. All of the bits you need to manipulate are in the `PT1AD1` register. This register can be accessed through the similarly named variable that has been declared in the `derivative.h` file. The PT1AD1 Register Diagram below should help you determine exactly which bits you need to read from or write to. Don't worry about why modifying certain bits will turn LEDs on and off. That will be your next topic in class.

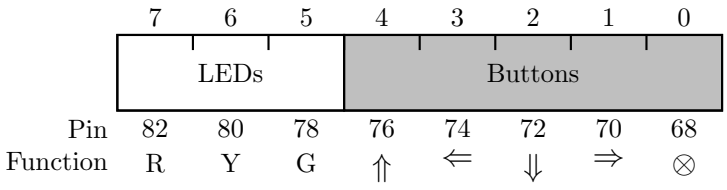

Figure 1: PT1AD1 Register Diagram

3.1 Create a new project

You should have a general understanding of how to set up a new project for an ICA at this point of the course. You should create a new branch of your git repository before creating the project. If you don't remember the git commands, there are several resources listed on Brightspace. As you are setting up your project, verify that these CodeWarrior Project Settings are correct.

Make the initial commit on your new branch once your project is configured and the provided starter code has been pasted into it.

Listing 1: CodeWarrior Project Settings

```
Processor = HCS12X - MC9S12XDP512
Connections = USBDM
Memory Model = Small
Floating Point = float is IEEE32, double is IEEE32
```

3.2 Initialize Port AD1

You haven't learned about configuring GPIO yet, but the starter code provided by your instructor contains the following Port AD1 initialization code. You will learn more about these lines in a later lecture, but they are configuring the pins connected to the LEDs as outputs and the pins connected to the buttons as inputs.

Listing 2: Port AD1 initialization

```
PT1AD1 &= 0b00011111;   //Turn off LEDs before enabling outputs
DDR01AD1 = 0b11100000;  //Set LED pins to be outputs
ATD1DIEN = 0b00011111;  //Disable A-to-D button pins (22.3.2.69)
```

Sets the register to completely new values overwriting the register and any original values will be lost

3.2.1 Part 2 Questions

This will turn top 5,6,7 bits while keeping the switches on

The first line in the code above is `PT1AD1 &= 0b00011111`. How is this different than `PT1AD1 = 0b00011111` or `PT1AD1 = 0b00000000`?

This is also overwriting the register but also setting the bits 0,1,2,3,4 off

3.3 Turn on two LEDs

If you refer to the PT1AD1 Register Diagram, you can see that bits 7 and 5 control the red and green LEDs. Add a line of code that will set those bits high without affecting the other bits of `PT1AD1` to the `One-time initializations` section. The red and green LEDs should illuminate when you run this code.

aPT1AD1 |= 0b10000000; on Red
PT1AD1 = 0b00100000; on Green

3.3.1 Part 3 Questions

How often will this line of code run?

on one time initialization so only once, if needed more than once run it inside the infinite loop

3.4 Toggle one LED

evert iteration just means to toggle it inside the infinite loop for(;;)

PT1AD1 ^= 0b01000000;

All of your code should now be added to the main loop of your code. Add a line of code that will toggle bit 6 of `PT1AD1` on evert iteration of the main loop without changing any other bits in the register. It should look like the yellow LED is illuminated when you run the code, but it will be less bright than the other two LEDs. Set a breakpoint on this line of code so you can see what is happening when it runs.

3.4.1 Part 4 Questions

How often does your new line of code run? infinite since it's inside of infinite loop
What operation does ^ perform? toggles the led on and off
Explain how ^ is making the LED toggle on an off. ^ is Xor meaning

3.5 Turn off one LED

Add another line that will clear bit 7 of `PT1AD1` without changing any other bits. The red LED should turn off when this line of code runs. If you let the main loop continue running, the yellow LED should still be toggling and the green LED should still be illuminated. Without any breakpoints, the yellow LED will be toggling too quickly for you to see; it will appear to be on continuously.

3.5.1 Part 5 Questions

Explain how the ~ operator is different from the ! operator.

The ~ operator is a bitwise NOT — it inverts every bit in a number, turning 1 to 0 and 0 to 1.

The ! operator is a logical NOT — it works on true/false values, turning true to false, and false to true. In C, !0 becomes 1, and anything non-zero becomes 0.

3.6 Bitmasking

If you refer to the PT1AD1 Register Diagram again, you will see that bits 0 - 4 are related to the buttons on your board. The LSB (least significant bit) shows the state of the middle button. To know if the button is being pressed, we need to be able to read the value of just that bit. Make a new `unsigned char` variable and assign it the value of PT1AD1 bit 0 using bitwise operations. Make sure this variable is getting updated on every iteration of the main loop.

Your new variable will be `true` if the center button is being pressed. Add an `if()` statement that will turn on the red LED if the button is being pressed. This should not change the state of the other two LEDs.

3.6.1 Part 6 Questions

You need to isolate a single bit to check if a specific button is being pressed because multiple buttons share the same register. Without isolating a bit, you can't tell which specific button (Up, Down, Left, Right, Center) is active.

Why do you need to isolate a single bit to check if a button is being pressed?

3.7 Bit shifting

Make sure you have made a *git commit* at this point. You can now reset your code to be the original starter code that was provided to you by your instructor. Simply copying and pasting the C file will be the simplest way to do this. You could use *git revert*, but you would need to close Code Warrior first.

Add the following Non-Blocking Delay code to the start of your main loop.

Listing 3: Non-Blocking Delay

```
static unsigned long delay = 0;
static unsigned char counter = 0;

if (++delay == 100000){ //When delay has incremented 100000 times
    counter++;  //Increment counter
    delay = 0;  //Reset delay timer
}
```

This will be covered in your next ICA, but it creates a short delay so that counter isn't constantly getting incremented. You now have a counter variable that will slowly be incremented as your code is running. We could use the LEDs to display the value in binary, but `PT1AD1 = counter;` would display the most significant bits (MSBs) of the counter variable on the LEDs. We want to see the LSBs on the LEDs. Use bit shifting operations to correctly display the counter value on the LEDs; make the three LSBs of the counter align with the three MSBs of `PT1AD1`.

3.7.1 Part 7 Questions

Why do you need to shift the counter values? What happens if you assign the counter to PT1AD1 without bit shifting? What happens to the LED display when the counter is greater than 7?

4 Conclusion

In this assignment you have seen several practical examples of bitwise operations. You will get plenty of practice with them throughout CMPE1250 because they are extremely common operations in embedded programming. Equally, many simple tasks will be very difficult until you have a strong understanding of these operations.