CPE 301 1104 - Lab Report 3

Lena Kemmelmeier, Lab Partner: Emma Cornia

March 1, 2024

Abstract

The purpose of this lab is to get familiarized with the GPIO. This is achieved by creating a simple circuit and program that takes input from a button and shows that input on one LED and the complement on the other LED. All input and output are taken using GIPO and bit masking.

1 Introduction

The objective of this lab was to gain a better understanding of General Purpose Input Output (GPIO), how the embedded system interacts with the microcontroller. A group of eight pins are grouped into a port, and we control GPIO pins with the DDR, PIN, and PORT registers.

The DDR (Data Direction Register) controls whether a pin is an input or output. The PORT register controls the state of a pin (whether it is high or low) if it is an output. If the pin is an input, then PORT controls whether or not the pullup resistor is enabled. PIN reflects the state of any pin. We use masking to manipulate the bits for these registers, using AND to set 0s and OR to set 1s.

2 Answers to Questions (Part 1)

2.1 Verify that the code is working – does the on-board Arduino LED blink? What is the frequency of the signal from the code?

Yes, the LED is blinking. The frequency of the signal from the code is 1 Hz (1/(500ms + 5000ms).

2.2 Modify the code in the example to blink faster, and check the output signal. What is the frequency of the signal?

The frequency of the signal from the code is 5 Hz (1/(100ms + 100ms).

2.3 Modify the example code to output the signal on both pin 13 and pin 7. (attach the part of the modified code here. You do not need to submit the ino file for this)

2.4 Connect an LED and a resistor in series to pin 7, and configure it such that the LED is on when pin 13 is driven high.

See Figure 3.

2.5 Connect an LED and a resistor in series to pin 7, and configure it such that the LED is off when pin 13 is driven low.

See Figure 4.

```
// Define Port B Register Pointers
volatile unsigned char* port_b = (unsigned char*) 0x25;
volatile unsigned char* ddr_b  = (unsigned char*) 0x24;
volatile unsigned char* pin_b  = (unsigned char*) 0x23;

void setup()
{
  //set PB7 to OUTPUT
  *ddr_b |= 0x80;
}

void loop()
{
  // drive PB7 HIGH
  *port_b |= 0x80;
  // wait 100ms instead of 500ms now
  delay(100);
  // drive PB7 LOW
  *port_b &= 0x7F;
  // wait 100ms instead of 500ms now
  delay(100);
}
```

Figure 1: Modified code for Part 1 so that the LED blinks faster.

```
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(7, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage level)
  digitalWrite(7, HIGH);
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the voltage LOW
  digitalWrite(7, LOW);
  delay(1000);                      // wait for a second
}
```

Figure 2: Modified code for Part 1 so that it outputs to both pins 7 and 13.

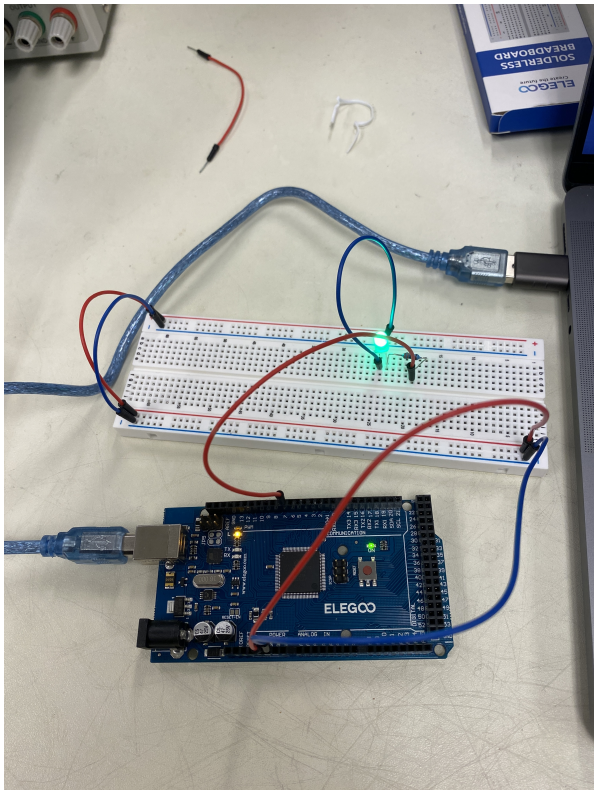

Figure 3: This is state 1, LED on when pin 13 driven high

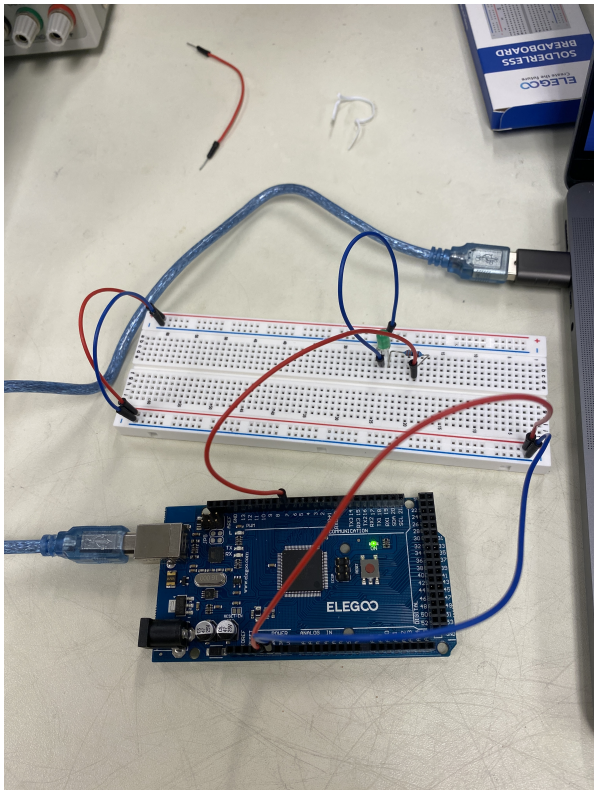

Figure 4: This is state 2, LED off when pin 13 driven high

3 Answers to Questions (Part 2)

3.1 What is the frequency of the signal on PB7 without modifying the code?

The frequency of the signal on PB7 is 1 Hz (1/(500ms + 500ms)).

3.2 What is the frequency of the signal on PB7 without modifying the code?

The fastest possible frequency is 500 Hz (1/(1ms + 1ms)).

3.3 What is the function of the line in the setup function? What does the DDR register do?

The line sets PB7 to output using a bit mask. The DDR register controls whether the pin is set to input or output.

3.4 Explain what the OR operation and the AND operation in the loop function are doing to the port register. What does the port register do?

The OR and AND operations in the loop function are bit masking the seventh bit of the port register to high and low. The port register controls the LED on the Arduino board.

```
#define WRITE_HIGH_PB(pin_num)  *port_b |= (0x01 << pin_num);
#define WRITE_LOW_PB(pin_num)  *port_b &= ~(0x01 << pin_num);

// Define Port B Register Pointers
volatile unsigned char* port_b = (unsigned char*) 0x25;
volatile unsigned char* ddr_b  = (unsigned char*) 0x24;
volatile unsigned char* pin_b  = (unsigned char*) 0x23;

void setup()
{
  //set PB7 to OUTPUT
  *ddr_b |= 0x80;
}

void loop()
{
  // drive PB7 HIGH
  WRITE_HIGH_PB(7);
  // wait 1ms instead of 500ms
  delay(1);
  // drive PB7 LOW
  WRITE_LOW_PB(7);
  // wait 1ms instead of 500ms
  delay(1);
}
```

Figure 5: Modified example 2 to have the greatest frequency output (shortest delay)

4 Answers to Questions (Part 3)

4.1 After compiling, the Arduino IDE lists the memory consumption for both program data, and variable data in the terminal window at the bottom of the IDE, compare the memory consumption of all three examples.

Example 1 uses 798 bytes of program storage space and 9 bytes of global variable space. Example 2 uses 858 bytes of program storage space and 9 bytes of global variable space. Example 3 uses 858 bytes of program storage space and 9 bytes of global variable space.

4.2 What are the frequencies achieved? Is there any difference in the maximum frequency output between examples 1 through 3? What explains the frequency differences?

The frequency achieved in example 2 is 500 Hz. The frequency achieved in example 3 is 500 Hz. There is no difference in the maximum frequency in example 1, example 2, and example 3. This is because they all use the delay function, which can take a minimum value of 1ms, which will make the frequency 500 Hz (1/(1ms + 1ms)).

5 Answers to Questions (Part 4)

5.1 Why do we need to enable the pull-up resistor on PB4? What would happen if the pull-up resistor was not enabled?

The pin is effectively disconnected without pull-up and may return a random value. If the pull-up resistor is not enabled, a random value may be given instead of the correct value.

5.2 Test your assumption from question 1 by modifying the example to disable the pull-up resistor. Is the input reliable? Does it reflect the state of the pin? What is the state of the pin when the button is not pressed?

When the pull-up resistor is not enabled, the input is not reliable and it does not reflect the state of the pin. When the button is not pressed the state of the pin is either high or low.

```
// Define Port B Register Pointers
volatile unsigned char* port_b = (unsigned char*) 0x25;
volatile unsigned char* ddr_b  = (unsigned char*) 0x24;
volatile unsigned char* pin_b  = (unsigned char*) 0x23;

void setup()
{
  //set PB7 to OUTPUT
  set_PB_as_output(7);
}

void loop()
{
  // drive PB7 HIGH
  write_pb(7, 1);
  // wait 1ms instead of 500ms
  delay(1);
  // drive PB7 LOW
  write_pb(7, 0);
  // wait 1ms instead of 500ms
  delay(1);
}
void set_PB_as_output(unsigned char pin_num)
{
    *ddr_b |= 0x01 << pin_num;
}
void write_pb(unsigned char pin_num, unsigned char state)
{
  if(state == 0)
  {
    *port_b &= ~(0x01 << pin_num);
  }
  else
  {
    *port_b |= 0x01 << pin_num;
  }
}
```

Figure 6: Modified example 3 to have the greatest frequency output (shortest delay)

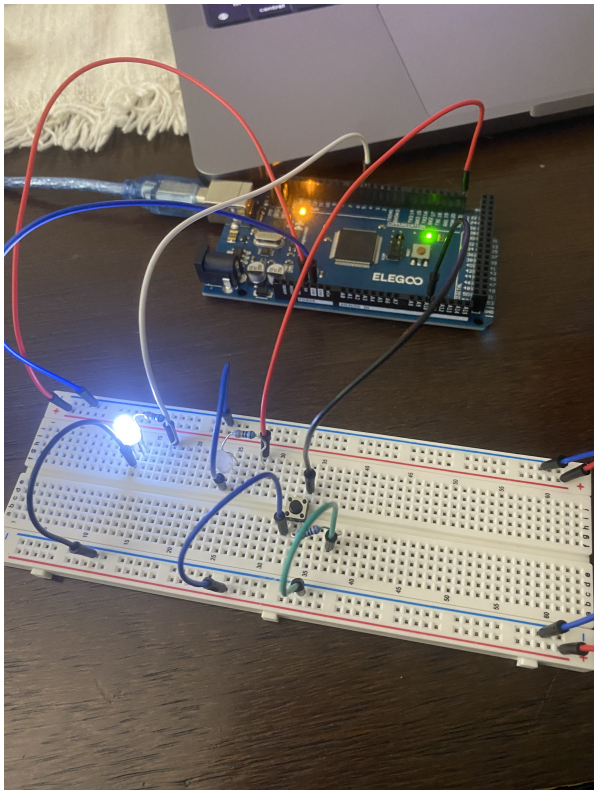

Figure 7: Results when button was not pressed

6 Experimental Design

This circuit includes 2 LEDs and 1 button. The LED on port PD0 will output the state of the button, connected to port PK2. The LED on port PE3 will output the complement of the state of the button. The code takes in the value of the button by using an AND bit mask with port PK2. If it is high, it sends a 1 to port PD0 using an OR bit mask and a 0 to port PE3 using an AND bit mask. The reverse is done when the state of the button is low. See Figure 7 to view the breadboard setup.

7 Results

When the button was not pressed, the LED connected to PE3 was turned on and the LED connected to PD0 was turned off. When the button was pressed, the LED connected to PE3 was turned off and the LED connected to PD0 was turned on. See Figures 7 and 8.

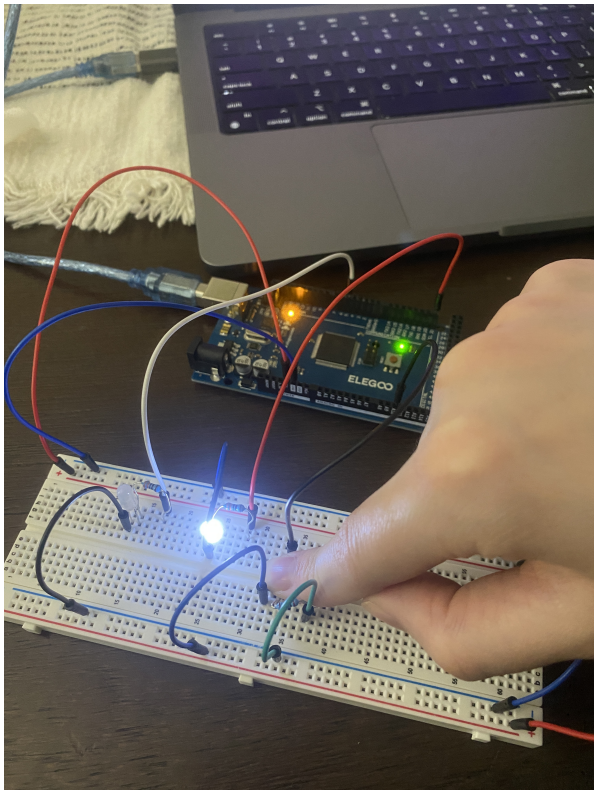

Figure 8: Results when button was pressed