GPIO Project 1

ECEN 220: Introduction to Embedded Systems
University of Nebraska–Lincoln
February 24, 2021

Name: David Perez

Contents

1 Introduction

The intention behind this project was to gain an understanding of the atmega328p GPIO (general purpose input output) registers. In the subsequent pages, there is source code and square waves to demonstrate how these registers behave in ways such as their execute time, creating delays, and even debouncing their pins.

2 Program Description

The purpose of program 1 is simply to introduce how to make a pin an output in order to allow an led to blink as fast as possible. The second programs intention was to print to the serial monitor each iteration that the loop was on and to take note of each instructions execution time. The third program on the other hand was nearly opposite of the first as its intended to create a delay that lasts X amount of milliseconds everytime the function is called. And finally, the final program introduces the concept of pin debouncing and how there is noise when reading signals and attempts to block out that noise via the delay function from program 3. Also, keep in mind that the source code for these programs will be located in the appendix section of this pdf.

2.1 Program 1

The loop is taking 9 cycles to complete which seems pretty reasonable to me. The frequency also seems fairly accurate because each period takes 2.7MHz / 375 ns to complete is 1/10 the clock speed that the MCU is running at.

The square wave is not showing a 50 percent duty cycle. I believe this is occurring because the led is off for a longer period than it is on which results from it having to return to the beginning of the loop.

If you wanted to create a 50 percent duty cycle you could insert a delay after you set PB1 high and with enough tinkering you could offset the lopsidedness that arises from the MCU having to return to the beggining of the loop.

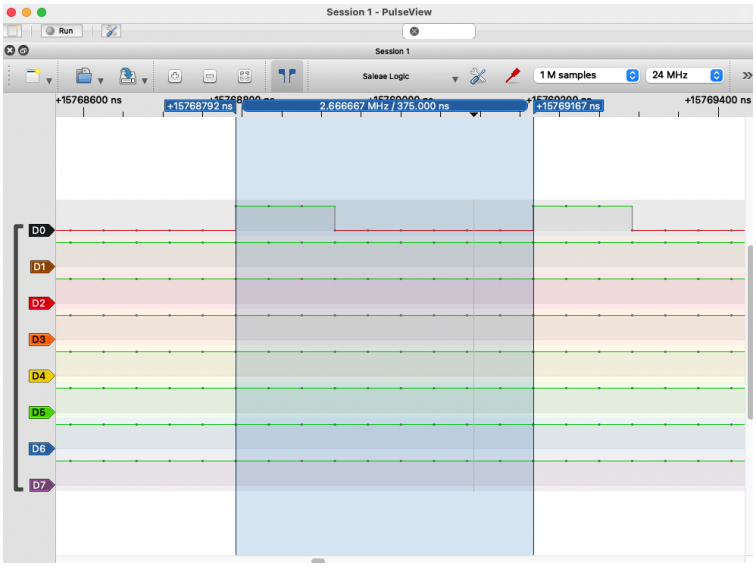

Figure 1: Screen Capture of program 1 square wave function

2.2 Program 2

As mentioned before the intent of this program was to caculate the execution time of some basic gpio commands on the atmega329p MCU. Figure 2 below shows that it took just about 20ms to set PB1 pin high, write a string to the serial monitor, then set PB1 low again. In figure 3, it displays a chunk of the code used to perform this action and how it was outputted onto the serial monitor.

Also take note that square wave shows that once the pin is turned high it takes roughly 19.77 ms to execute the sprintf and Serial.write functions. On the contrary it only takes about 1 microsecond to turn the pin low.

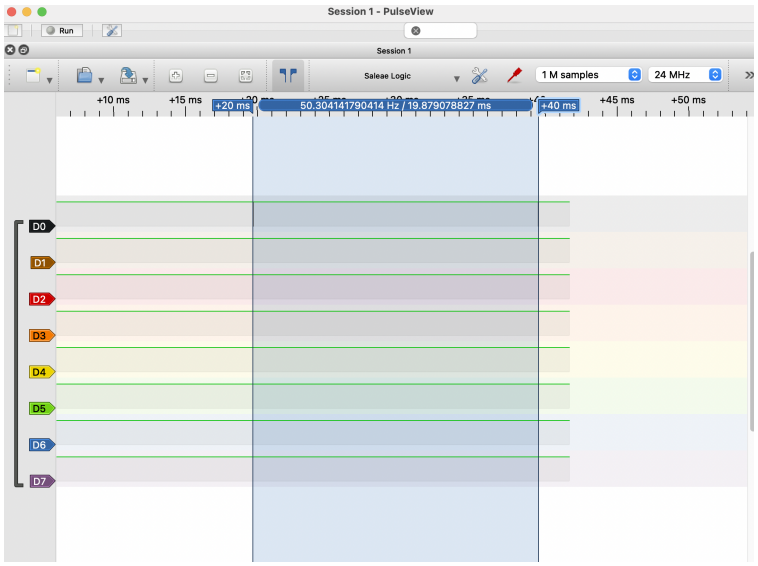

Figure 2: Screen Capture of program 2 square wave function

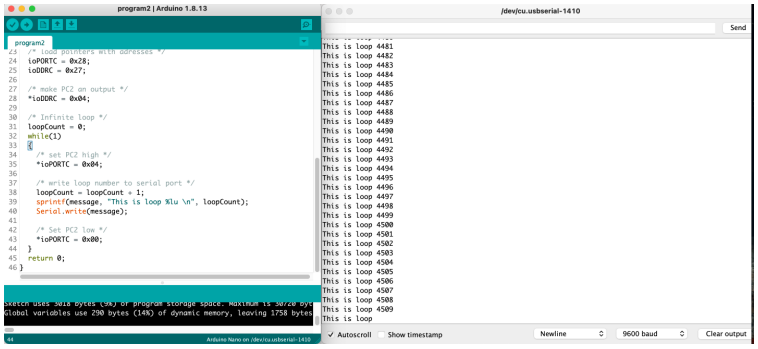

Figure 3: Screen Capture of program 2 square wave function

Another important thing to understand is that executing the serial functions are very time consuming for the MCU. When adding only 9 characters onto the message string it took the MCU another 10 ms totaling roughly 30.18 ms versus the 19.77 ms without the extra characters. This is shown in figure 4.

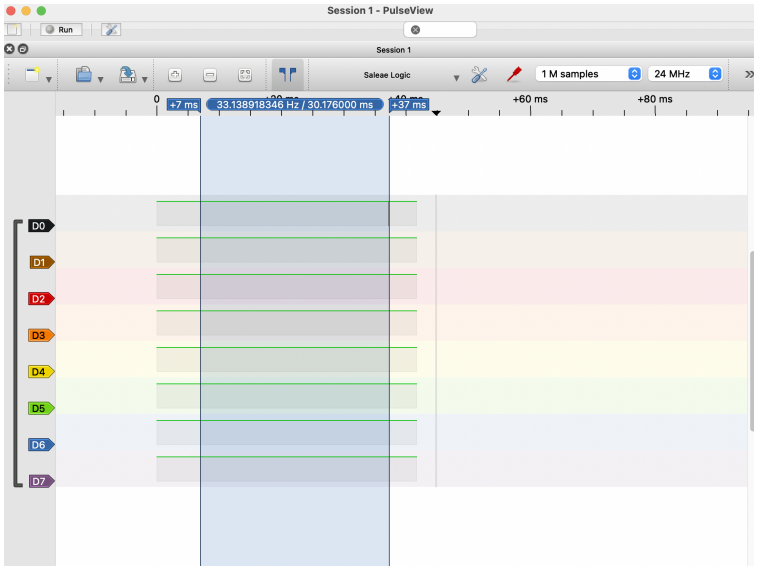

Figure 4: Screen Capture of program 2 delayed square wave

2.3 Program 3

In order to get a proper millisecond delay function I used a value of 373 that I would then multiply by the arguments passed into my "myHardDelay" function.

The value that resulted from the multiplication of 373 and the parameter, was passed into a variable called delayCount that was made volatile in order to prevent the delay from being optimized by the compiler.

When “1” is inserted into the function, the period (from rising edge to falling edge) is 2.016 ms. Thus the delay is roughly 1ms, that is 1ms when the pin is turned high and another 1ms when the delay is called again, after turning the pin low . When it comes to inserting 50 into the function the delay gets only slightly more inaccurate. The delay function took 50.15ms to execute when 50 was an argument to the function.

I found that the maximum delay would be roughly 5,757,329 which I derived from taking the maximum decimal digit a 32 bit number could hold then dividing by my N which is 373.

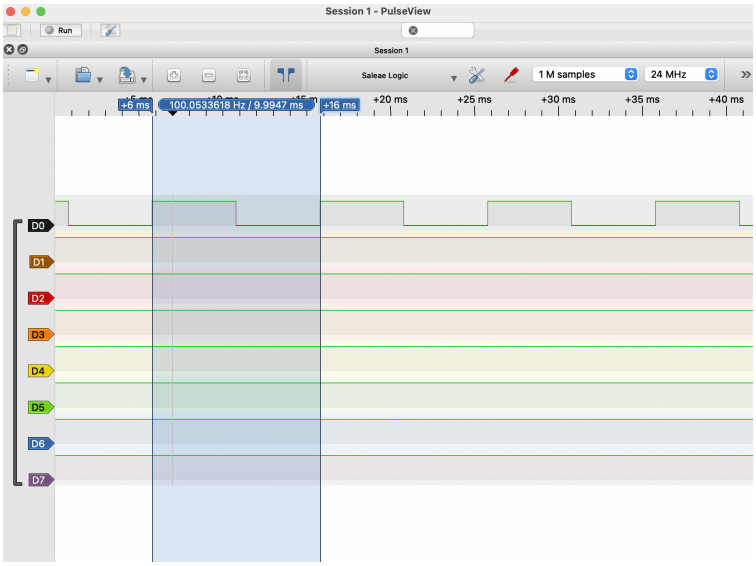

Figure 5: Screen Capture of program 3 100Hz square wave

2.4 Program 4

Without debouncing the pin it was apparent that there was indeed a need to debounce pins especially in the case of the push button. Without a delay signal read, the serial monitor would display presses even after the button had already been released.

The delay I chose is roughly 68 milliseconds. I chose this value because I had to take into account the time it took for the serial monitor to actually output my stings. If I had increased that delay too much more I'd also run into the problem of the serial monitor showing a button not pressed string when the button was pressed and would even exclude the bouncing signal.

I found 68 milliseconds to be just about the perfect delay where it would display the proper status of the button press. When decreasing only a few milliseconds the serial monitor would show multiple lines declaring the button was pressed when it wasn't.

3 Conclusion

Through each program there were some very good fundamental takeaways. From the first program one could gain an understanding of how gpio registers work. Continuining to program 2 I was able to understand the impact that certain commands have and ones like the Arduino's "Serial.write" take significantly longer than simply changing a register value. An important takeaway from program 3 is simply the architecture of how a delay function is wrote. Combining the delay in program 3 with the debouncing pins and with program 4 you could accuratley read changing values from a simple small electronic button.

4 Appendix

```c
#include <stdint.h>

/** GLOBAL Variables */
volatile uint8_t* pDDRB;
volatile uint8_t* pPORTB;


/* Main Function */

int main(void)
{
  /* Define our pointers to the GPIO registers */
  pDDRB = 0x24;
  pPORTB = 0x25;

  /* Set PB1 as an output with the DDRB register */
  *pDDRB = 0x02; // 0x01 = 0b00000010

  while(1)
  {
    /* Set PB1 high */
    *pPORTB = 0x02;

    /* Clear PB1 low */
    *pPORTB = 0x00;

  }
}
```

Listing 1: Program 1

```c

  /** Includes **/
  // arduino wiring library (contains serial.begin() and serial.write()
  #include <Arduino.h>
  #include <stdint.h>
  #include <stdio.h>  // contains sprintf() and snprintf()

  /** Global Variable **/
  volatile uint8_t* ioPORTC;
  volatile uint8_t* ioDDRC;
  uint32_t loopCount;
  char message[80];

  int main()
  {
    /* initialize the arduino wiring library */
    init();

    /* inintialize the serial port with a band of 9600 bits per second */
    Serial.begin(9600);

    /* load pointers with adresses */
    ioPORTC = 0x28;
    ioDDRC = 0x27;

    /* make PC2 an output */
    *ioDDRC = 0x04;

    /* Infinite loop */
    loopCount = 0;
    while(1)
    {
      /* set PC2 high */
      *ioPORTC = 0x04;
```

```
35
36        /* write loop number to serial port */
37        loopCount = loopCount + 1;
38        sprintf(message, "This is loop %lu \n", loopCount);
39        Serial.write(message);
40
41        /* Set PC2 low */
42        *ioPORTC = 0x00;
43      }
44      return 0;
45    }
```

Listing 2: Program 2

```
1
2   #include <stdint.h>
3
4   volatile uint32_t i;
5   volatile uint8_t* pDDRB;
6   volatile uint8_t* pPORTB;
7   volatile uint32_t delayCount;
8
9   void myHardDelay(uint32_t delayInMsec);
10
11  int main()
12  {
13
14    //load pointers with addresses
15    pDDRB = 0x24;
16    pPORTB = 0x25;
17
18
19    //make PB1 an output
20    *pDDRB = 0x02; // 0x01 = 0b00000010
21
22
23    while(1)
24    {
25
26      //set PB1 high then delay for 1 milliseconds
27      *pPORTB = 0x02;
28
29      //delay 1 millisecond
30      myHardDelay(1);
31
32      //set PB1 low then delay for 1 milliseconds
33      *pPORTB = 0x00;
34      myHardDelay(1);
35    }
36  }
37  void myHardDelay(uint32_t delayInMsec)
38  {
39    delayCount = delayInMsec * 373;
40    for(i=0;i <delayCount; i++){
41    }
42  }
```

Listing 3: Program 3

```
1
2   /* Includes */
3   #include <Arduino.h>
4   #include <stdint.h>
5   #include <stdio.h>
6
7   /** GLOBAL Variables */
8   volatile uint8_t* pPINB;
```

```
9  volatile uint8_t* pDDRB;
10 volatile uint8_t* pPORTB;
11
12
13
14
15 /* Function Declarations */
16 int8_t debouncePin(volatile uint8_t* mmrPINx, uint8_t bitToRead);
17 void myHardDelay(uint32_t delayInMsec);
18
19
20 // Main Function
21 int main(void)
22 {
23   // Initialize the Arduino Wiring Library
24   init();
25   //Initialize the serial port with a band of 9600 bits per second
26   Serial.begin(9600);
27
28   /* Define our pointers to the GPIO registers */
29   pPINB = 0x23;
30   pDDRB = 0x24;
31   pPORTB = 0x25;
32
33   //Initialize Variables
34   uint8_t pinState;
35
36   // set PB0 as an input
37   *pDDRB = 0x00;  //0b00000001
38
39   // Set PB0 as having a internal pullup resistor enabled
40   *pPORTB = 0x01; //0b00000001
41
42   //Read PB0 first
43   pinState = *pPINB & 0x01; // 0b00000001
44
45   while(1)
46   {
47     // Check the debounce state from my debounce function
48     int8_t state = debouncePin(0x23, 0x01);
49
50     if (state == 0)
51     {
52       // Pin is bouncing
53       Serial.write("The pin is bouncing\n");
54     }
55     else if (state == 1)
56     {
57       // Pin is debounced high
58       Serial.write("The pin is being pushed\n");
59     }
60     else if (state == -1)
61     {
62       // Pin is debounced low
63       Serial.write("The pin is NOT being pushed\n");
64     }
65   }
66 }
67
68 int8_t debouncePin(volatile uint8_t* mmrPINx, uint8_t bitToRead)
69 {
70   //read PB0
71   uint8_t firstDebouceSample = *mmrPINx & bitToRead;
72   //Delay for 67 milliseconds
73   myHardDelay(67);
74
75   uint8_t secondDebouceSample = *mmrPINx & bitToRead;
76
```

```
77    if(firstDebouceSample == secondDebouceSample)
78    {
79      //The pin is successfully debounced
80      if(firstDebouceSample == 0x00)
81      {
82        //The pin is being pushed (debounced and high)
83        return 1;
84      }
85      else
86      {
87        // The pin is not being pushed (debounced and low)
88        return -1;
89      }
90    }
91    else{
92      // The pin is bouncing
93      return 0;
94    }
95  }
96
97  void myHardDelay (uint32_t delayInMsec)
98  {
99    volatile uint32_t i;
100   uint32_t delayCount = delayInMsec * 373;
101   for(i=0;i <delayCount; i++){
102     //do nothing
103   }
104 }
```

Listing 4: Program 4