Lab01 - GPIO

Introduction to Embedded Systems - University of Nebraska

Dat Nguyen

Contents

1 Introduction

This lab provides hand-on activity and insight of General Purpose Input/Output (GPIO) usages.

2 Program Description

2.1 Program 1 - Period of The Square Wave

Intuitively, one would expect to see an 8 MHz square wave with an assumption that each C instruction on line 17 and line 19 in Appendix 4.1 would take 1 clock cycle each. However, this is a wrong assumption as the resulted square wave is roughly 2.6 MHz. This indicate that the two C instruction take roughly 6 instructions in total.

Figure 1 show the screen captured signal on PB1. Notice that the "on" duty cycle of the square wave is shorter than "off" cycle. After PB1 is set to low, the program counter required extra cycle to go back up the while loop to set the pin high again , thus, the pin stay low during the extra cycle time.

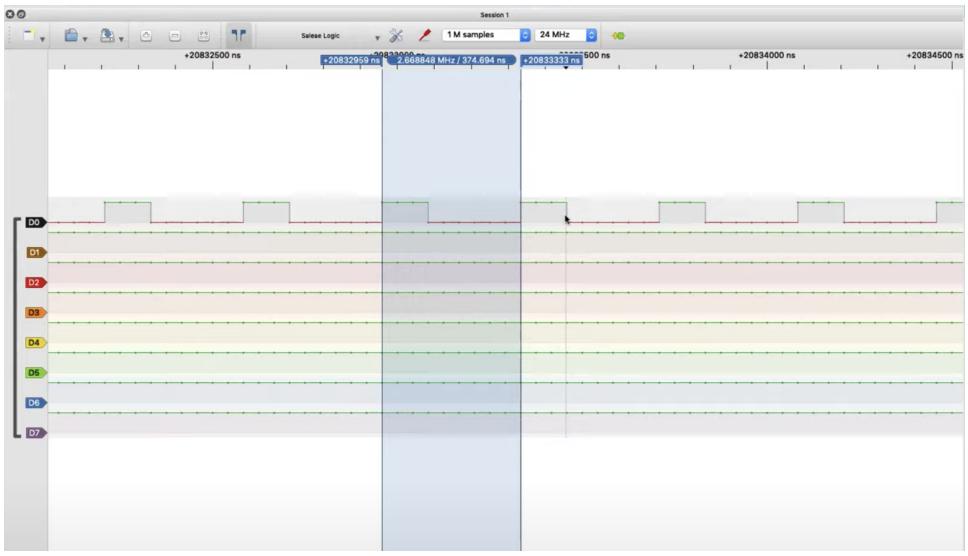

Figure 1: Screen Capture of Signal on PB1

2.2 Program 2 - Serial.Write Execution Time

For message length of 1, the high period of pin PB1 is 1ms. The low period is roughly the same as high period because the same message is printed after pin is set to low. From this result, one can conclude that the time it take to "serial write" the run time debug message of length 1 is roughly 1ms subtracting the amount of time it take to turn pin PB1 to low. The corresponding square wave frequency is 1 KHz. In one cycle, two message of length one will be printed to balance out the high and low duty cycle. Figure 2 show PB1 signal when printing message of length 1.

For message length of 10, the high period is 10.4ms with the corresponding square wave frequency of 96 Hz. Noticing the changes when printing message of length 1 and 10, we can conclude that the length of the message is proportional with time to print them.

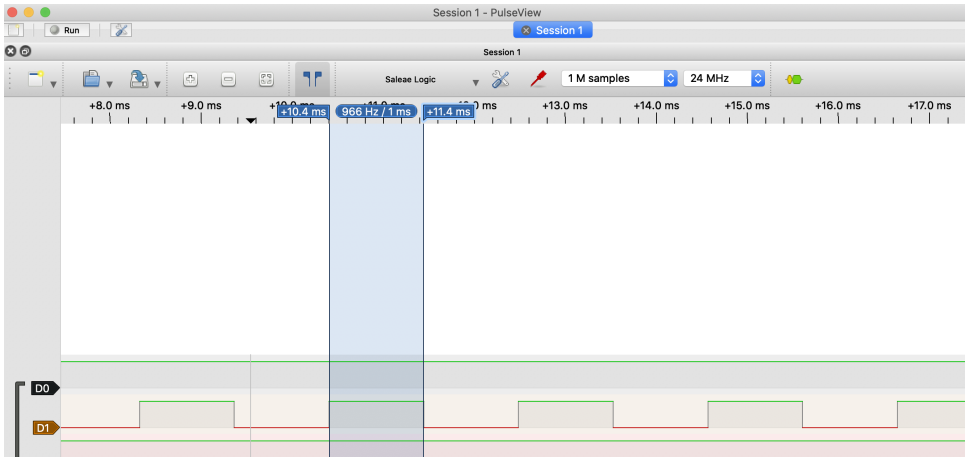

Figure 2: show PB1 signal when printing message of length 1

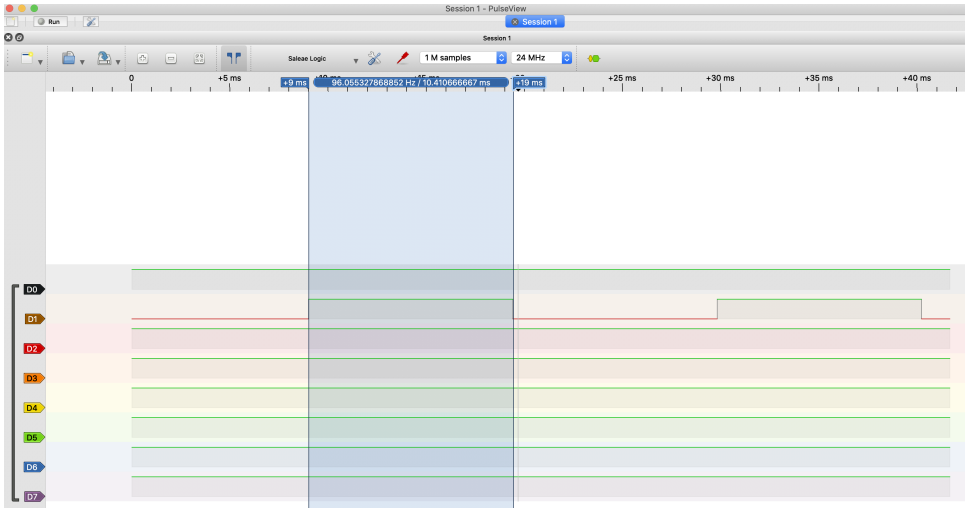

Figure 3: show PB1 signal when printing message of length 10

2.3 Program 3 - Custom Delay

The N value is found to be 835. When executing the function to delay the program for 1ms, the delayed value fluctuate between 999.33μs and 1000.666μs. The delay function at 50ms is 49.96ms. This is expected because the fluctuation is truly random therefore, the error relatively the same when the program is delayed for 1ms or 50ms. The delay function take in a maximum value of $2^{32} - 1$ represent the maximum value of $uint32\_t$ for the ms input parameter. The delay function is implemented as following:

```
1 void myHardDelay(uint32_t ms) {
2   volatile int16_t count;
3
4   while (ms) {
5     for (count = 0; count < 835; count++);
6     ms -= 1;
7   }
8 }
```

Variable ms is passed in as total time that the program will be delayed in milli seconds.

The program will be stall in the while loop until ms count reaching 0.

Figure 4 screen capture display the Arduino creating a 100Hz square wave, corresponding with 10ms delay between each pin toggle, out of PB1.

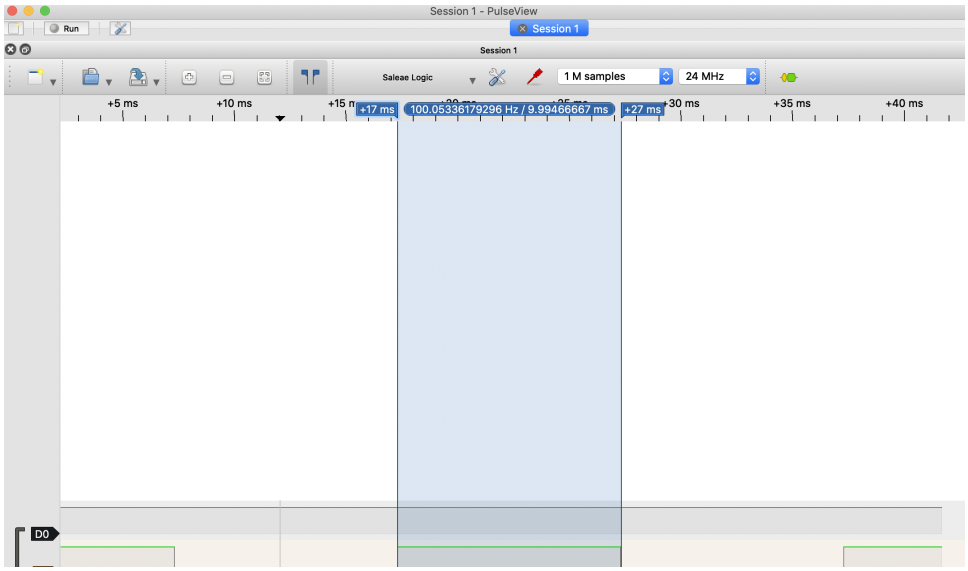

Figure 4: Show 100 Hz Square wave from PB1

2.4 Program 4 - Debouncing

1. The debounce function was sanity checked by add logic to recheck if the pin reading is opposite of the current pin reading value after the first read. For example, when checking for reaching equals zero then the button input pin should not read 1 right after. Below snippet shows logic to detect if button bouncing happens when pin is pressed.

```
1     if (reading == 0) {
2       Serial.write("Pressed\n");
3       if ((*ioPINB & 0x01) == 1) {
4         Serial.write("Released while pressing");
5       }
6     }
```

2. With a delay of 20ms between each pin reading, button bouncing rarely happens. To ensure the false button reading, a delay of 40ms was added between each pin reading.

3. If decreasing the time delay between pin reading to 25ms, we encounter situation of false reading rarely, approximately 1 false reading every 10 or 20 button pressed.

3 Summary

This lab emphasized following important key points

1. The mapping between C operations don't translate directly to assembly instructions in term of clock cycle.

2. UART I/O is expensive where each character can take up to 1ms to be printed.

3. A custom delay can be implement by stalling the program counter at a position translating to a blank loop in C within a certain limit

4. Button is a cheap mechanical device with bouncing issue, where pin pressed can sometime falsely reported, therefore, a delay between each pin reading should be added to avoid this problem.

4 Appendix

4.1 Program 1

```
1  #include <stdint.h>
2
3  //global
4  volatile uint8_t *ioPORTB;
5  volatile uint8_t *ioDDRB;
6
7  int main()
8  {
9    ioPORTB = (uint8_t *)0x25;
10   ioDDRB = (uint8_t *) 0x24;
11
12   //make PB1 as output
13   *ioDDRB = 0x02; // DDRB[7:0] 0000 0010
14
15   while (1) {
16     // set PB1 high
17     *ioPORTB = 0x02; // 0000 0010
18     // set PB1 low
19     *ioPORTB = 0x00; // 0000 0000
20   }
21 }
```

4.2 Program 2

```
1  #include <stdint.h>
2
3  //global
4  volatile uint8_t *ioPORTB;
5  volatile uint8_t *ioDDRB;
6
7  const int msgLen = 1;
8  uint8_t msg[msgLen+1];
9
10 int main()
11 {
12   init(); // init arduino libraries
13   Serial.begin(9600);
14
15   ioPORTB = (uint8_t *)0x25;
16   ioDDRB = (uint8_t *) 0x24;
17
18   //make PB1 as output
19   *ioDDRB = 0x02; // DDRB[7:0] 0000 0010
20
21   for (int i=0; i<msgLen; i++) {
22     msg[i] = 'r';
23   }
24   msg[msgLen] = '\0';
25
26   uint32_t count = 0;
```

```
27    while (1) {
28      // set PB1 high
29      *ioPORTB = 0x02; // 0000 0010
30
31      Serial.write((char *) msg);
32
33      // set PB1 low
34      *ioPORTB = 0x00; // 0000 0000
35      Serial.write((char *) msg);
36    }
37  }
```

4.3 Program 3

```
1   #include <stdint.h>
2
3   //global
4   volatile uint8_t *ioPORTB;
5   volatile uint8_t *ioDDRB;
6
7
8   void myHardDelay(uint32_t ms) {
9     volatile int16_t count;
10
11    while (ms) {
12      for (count = 0; count < 835; count++);
13      ms -= 1;
14    }
15  }
16
17  int main()
18  {
19    init(); // init arduino libraries
20    Serial.begin(9600);
21
22    ioPORTB = (uint8_t *)0x25;
23    ioDDRB = (uint8_t *) 0x24;
24
25    //make PB1 as output
26    *ioDDRB = 0x02; // DDRB[7:0] 0000 0010
27
28
29    while (1) {
30      // set PB1 high
31      *ioPORTB = 0x02; // 0000 0010
32
33      myHardDelay(1);
34
35      // set PB1 low
36      *ioPORTB = 0x00; // 0000 0000
37      myHardDelay(1);
38    }
39  }
```

4.4 Program 4

```
1   #include <stdint.h>
2
3   //global
4   volatile uint8_t *ioPORTB, *ioDDRB, *ioPINB;
5
6   void myHardDelay(uint32_t ms) {
7     volatile int16_t count;
```

```
8
9     for (;ms > 0; ms--) {
10      for (count = 0; count < 835; count++);
11    }
12  }
13  int main()
14  {
15    init(); // init arduino libraries
16    Serial.begin(9600);
17
18    ioPORTB = (uint8_t *)0x25;
19    ioDDRB = (uint8_t *) 0x24;
20    ioPINB = (uint8_t *) 0x23;
21
22    *ioDDRB = 0x00; // // make port B become input including DDRB[0]
23    *ioPORTB = 0x01; // Use input pull up on PB0
24
25    while (1) {
26      uint8_t reading = (*ioPINB & 0x01);
27
28      if (reading == 0) {
29        Serial.write("Pressed\n");
30        if ((*ioPINB & 0x01) == 1) {
31          Serial.write("Released while pressing");
32        }
33      } else {
34        Serial.write("Released\n");
35        if ((*ioPINB & 0x01) == 0) {
36          Serial.write("Pressed while releasing");
37        }
38      }
39      myHardDelay(40);
40    }
41  }
```