Lab 3: Bit manipulation

UCR EE/CS 120B

Pre-lab

Read the introduction and write a `tests.gdb` for each exercise 1, 2, and 3 targeting the ATmega1284.

Introduction

This lab will cover further exploration with the AVR toolchain. Manipulating bits is an important aspect of programming embedded systems (while less necessary when programming desktop systems). For example:

```
#include <avr/io.h>
#ifdef _SIMULATE_
#include "simAVRHeader.h"
#endif

int main(void)
{
	DDRA = 0x00; PORTA = 0xFF; // Configure port A's 8 pins as inputs, initialize to 1s
	DDRB = 0xFF; PORTB = 0x00; // Configure port B's 8 pins as outputs, initialize to 0s
	unsigned char tmpB = 0x00; // You are UNABLE to read from output pins. Instead you
	// Should use a temporary variable for all bit manipulation.
	unsigned char button = 0x00;
	while(1)
	{
		// 1) Read input
		button = PINA & 0x01;
		// 2) Perform Computation
		// if PA0 is 1, set PB1PB0 = 01, else = 10
		if (button == 0x01) { // True if PA0 is 1
			tmpB = (tmpB & 0xFC) | 0x01; // Sets tmpB to bbbbbb01
						// (clear rightmost 2 bits, then set to 01)
		}
		else {
			tmpB = (tmpB & 0xFC) | 0x02; // Sets tmpB to bbbbbb10
						// (clear rightmost 2 bits, then set to 10)
		}
		// 3) Write Output
		PORTB = tmpB; // Sets output on PORTB to value of tmpB
	}
}
```

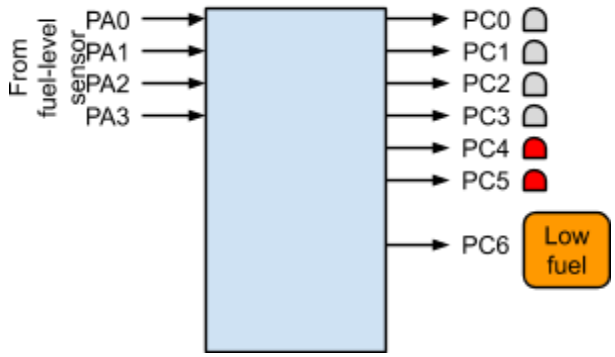
Note: A handy debug tactic is to use the display list. Any variable you define can be added to the display list. To do so, build your code with the `-g` and `-Og` flags, then add the variable using the command "`display /FMT <variable>`" where `/FMT` can be hexadecimal (x), decimal (d), or binary (t) depending on what you are trying to visualize.

As you step through the code you can then keep track of the value of your variables, such as '`button`' in the example above. Once you have stepped past the line of code and it has executed, any changes to your variables will be displayed.

Exercises

Write C programs for the following exercises targeting the ATmega1284. These behaviors are combinational; you do not need state machines. Demo each part to a TA.

1. Count the number of 1s on ports A and B and output that number on port C.
   ```
   $ cp source/main.c turnin/[cslogin]_lab3_part1.c
   $ cp test/tests.gdb turnin/[cslogin]_lab3_part1_tests.gdb
   $ git commit -m "Completed part 1"
   ```
2. A car has a fuel-level sensor that sets PA3..PA0 to a value between 0 (empty) and 15 (full). A series of LEDs connected to PC5..PC0 should light to graphically indicate the fuel level. If the fuel level is 1 or 2, PC5 lights. If the level is 3 or 4, PC5 and PC4 light. Level 5-6 lights PC5..PC3. 7-9 lights PC5..PC2. 10-12 lights PC5..PC1. 13-15 lights PC5..PC0. Also, PC6 connects to a "Low fuel" icon, which should light if the level is 4 or less. (The example below shows the display for a fuel level of 3).




   ```
   $ cp source/main.c turnin/[cslogin]_lab3_part2.c
   $ cp test/tests.gdb turnin/[cslogin]_lab3_part2_tests.gdb
   $ git commit -m "Completed part 2"
   ```
3. In addition to the above, PA4 is 1 if a key is in the ignition, PA5 is one if a driver is seated, and PA6 is 1 if the driver's seatbelt is fastened. PC7 should light a "Fasten seatbelt" icon if a key is in the ignition, the driver is seated, but the belt is not fastened.
   ```
   $ cp source/main.c turnin/[cslogin]_lab3_part3.c
   $ cp test/tests.gdb turnin/[cslogin]_lab3_part3_tests.gdb
   $ git commit -m "Completed part 3"
   ```
4. (**Challenge**): Read an 8-bit value on PA7..PA0 and write that value on PB3..PB0PC7..PC4. That is to say, take the upper nibble of PINA and map it to the lower nibble of PORTB, likewise take the lower nibble of PINA and map it to the upper nibble of PORTC (PA7 -> PB3, PA6 -> PB2, … PA1 -> PC5, PA0 -> PC4).
   ```
   $ cp source/main.c turnin/[cslogin]_lab3_part4.c
   $ cp test/tests.gdb turnin/[cslogin]_lab3_part4_tests.gdb
   $ git commit -m "Completed part 4"
   ```

5. (**Challenge**): A car's passenger-seat weight sensor outputs a 9-bit value (ranging from 0 to 511) and connects to input PD7..PD0PB0 on the microcontroller. If the weight is equal to or above 70 pounds, the airbag should be enabled by setting PB1 to 1. If the weight is above 5 but below 70, the airbag should be disabled and an "Airbag disabled" icon should light by setting PB2 to 1. (Neither B1 nor B2 should be set if the weight is 5 or less, as there is no passenger).

```
$ cp source/main.c turnin/[cslogin]_lab3_part5.c
$ cp test/tests.gdb turnin/[cslogin]_lab3_part5_tests.gdb
$ git commit -m “Completed part 5”
$ git push
```

Note: A port can be set to be input for some pins, and output for other pins by setting the `DDR` appropriately, e.g. `DDRA = 0xF0;` will set port A to output on the high nibble, and input on the low nibble.

Submission

Each student must submit their source files (`.c`) and test files (`.gdb`) according to instructions in the lab submission guidelines.

```
$ tar -czvf [cslogin]_lab2.tgz turnin/
```

Don’t forget to commit and push to Github before you logout!