UCR EE/CS 120B

Lab 2: Bit manipulation (Vehicle sensors)

Don't forget to create the new project in your Github desktop client.

This lab will cover further exploration with Atmel Studio 7. Manipulating bits is an important aspect of programming embedded systems (while less necessary when programming desktop systems). Furthermore, making the code clear and readable is important. A good practice is to 1) read Inputs and assign to variables, 2) perform the necessary computation, and 3) write results to port. For example, the following code from an earlier lab:

```
#include <avr/io.h>

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
	}
}
```

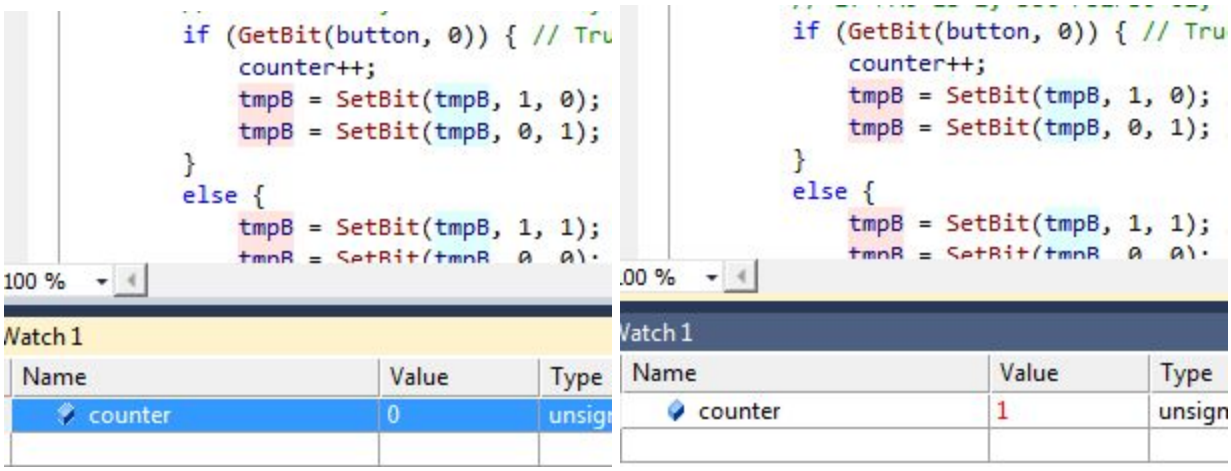
might be better written as:

```
#include <avr/io.h>

// Bit-access function
unsigned char SetBit(unsigned char x, unsigned char k, unsigned char b) {
       return (b ? x | (0x01 << k) : x & ~(0x01 << k));
}
unsigned char GetBit(unsigned char x, unsigned char k) {
       return ((x & (0x01 << k)) != 0);
}

int main(void)
{
       DDRA = 0x00; PORTA = 0xFF; // Configure port A's 8 pins as inputs
       DDRB = 0xFF; PORTB = 0x00; // Configure port B's 8 pins as outputs,
                                                                  // initialize to 0s
       unsigned char tmpB = 0x00; // intermediate variable used for port updates
       unsigned char button = 0x00;
       while(1)
       {
              // 1) Read Inputs and assign to variables
              button = PINA & 0x01; // Mask PINA to only get the bit you are interested in
              // 2) Perform Computation
              // if PA0 is 1, set PB1PB0=01, else =10
              if (GetBit(button, 0)) { // True if PA0 is 1
                     tmpB = SetBit(tmpB, 1, 0); // Set bit 1 to 0
                     tmpB = SetBit(tmpB, 0, 1); // Set bit 0 to 1
              }
              else {
                     tmpB = SetBit(tmpB, 1, 1); // Set bit 1 to 1
                     tmpB = SetBit(tmpB, 0, 0); // Set bit 0 to 0
              }
              // 3) write results to port
              PORTB = tmpB;
       }
}
```

Note: SetBit() and GetBit() are useful functions when manipulating 1 or 2 bits, when manipulating more bits at a time it is easier, and cleaner, to use bit manipulation techniques such as masking and shifting.

Note: A handy debug tactic is to use the watch list. Any variable you define can be added to the watch list. To do so, build your code, enter debug mode and 'break all'. You can then right click on your variables and add them to the watch list. 

As you step through the code you can then keep track of the value of your variables, such as 'counter' in the example below. Once you have stepped past the line of code and it has executed, any changes to your variables will be marked in red. Additionally you can manually edit the watch list values, which will change the current value of the variable within the program's execution. This is handy to test loop and branching conditions.



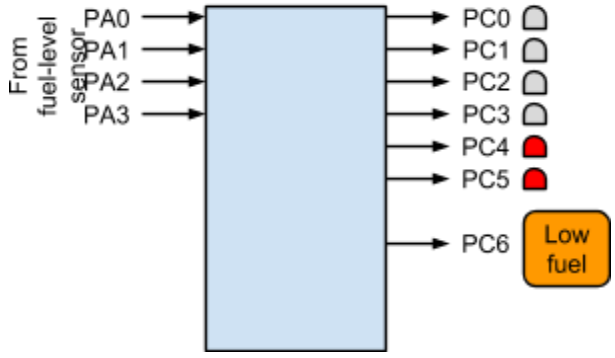
Pre-lab

Read the above and write the main C code for exercises 1 and 2 targeting Atmel Studio.

Exercises

Write C programs for the following exercises using Atmel Studio for an ATMega1284. Use GetBit or SetBit functions **as appropriate** (See Note above). 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.
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).



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.
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).

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).

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.

Each student must submit their .c source files according to instructions in the lab submission guidelines.

Don’t forget to commit and push to Github before you shutdown the VM!