LAB 3
First steps with a micro-controller
Input Output voltage levels, Loop Timing, and
Connecting LEDs and Switches

Suggested reading: **What is A Microcontroller, A Student Guide**, by Parallax, Inc., Chapters 1, 2 and 3. The Physics 430 experiments cover somewhat the same material as the Student Guide, but in a different manner and faster. Reading the Parallax documentation before starting might be useful. Sometimes more detailed instructions are given in that guide and additional tricks are shown. ((WAMC2_1.pdf available from Parallax web site http://www.stampsinclass.com or on the Stamp CD.)

1. Pre-flight Hardware check

Check for the following:

Basic Stamp is plugged into the experimental board properly.

Check for correct orientation. Is it pushed in all the way?

Are there any bent pins?

RS232 cable is connect to the Stamp's serial interface and to the computer's serial I/O port.
[If your workstation has no RS232 but has USB, then there needs to be a USB-RS232 adapter.]
A 9 V power supply or battery is plugged into the power connections.
(make sure the switch is on "0" before connecting the power.)

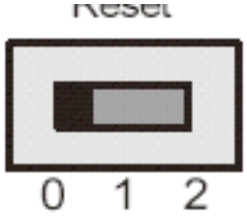

Fig 1: The three-position off/on switch. Position 0 is off. Position 1 is for normal operation without servo motors and position 2 is for powering the servo motors.

2. Pre-flight software and interconnection check

Start the Stamp Editor on your workstation.

Turn the power switch on the stamp to "1"

Click the identify icon in the stamp editor and check that the Editor recognizes that the stamp is connected and that it responds correctly. The name of the stamp and firmware version should be displayed on the editor's screen.

If this doesn't work, there is something wrong that needs correcting before you continue.

3. First Program

Load or type the first test program into the editor. It's called FirstProgram.bs2.
It should be similar to the following one.

```
' What's a Micro-controller - FirstProgram.bs2
' BASIC Stamp sends message to Debug Terminal.
' {$STAMP BS2}
' {$PBASIC 2.5}

DEBUG CLS, "Hello World!", CR
```

```
END
```

```
[Program notes:
The lines beginning with a ' are comments and are ignored by the Stamp. They
aren't even downloaded so you can make as many comments as you need without
worrying about using up memory in the Stamp.  Two special comments in the
curly braces {} are called directives. The first one specifies which version
of the Stamp microprocessor you're using and the second which version of
PBASIC language the code is written in. You should always include both
directives because I'm not sure what would happen if you don't. The software
might make an assumption about the Stamp model or language version which is
incorrect, or it might not work at all.]
```

Record the results of this test in your notes.

4. Voltage Levels

Before going on to the next exercise please consider the following caution: Be very careful to only touch one pin at a time when you make measurements with your probe. If you feel unsteady, connect a wire to the output pin that you're measuring, while the power is off, and clip the probe onto the wire so that an accidental twitch or slip won't cancel your Stamp by connecting two pins which shouldn't be connected. (eg, Vin to P15)

Examine and record the voltage levels on Vss, Vdd and Vin.

Fig 2: The end of the waffle board has a strip with the following connections: Vdd, regulated 5 V, Vin, unregulated power from the supply or battery, and Vss, ground.

Examine and record the voltage levels on the output pins after the "Hello World" program has been downloaded.

Question: The "hello world" program does not make any use of the I/O pins. In this case the I/O pins are supposed to be in input mode and they exhibit a high impedance to anything connected to them. Because the voltmeter has over a MΩ input impedance, you cannot tell from the measured voltage whether the pins are high impedance or at zero volts. Can you devise a way to distinguish the high impedance state from a normal high or low output? (If this question doesn't make any sense to you then you don't understand 3-state outputs. Ask an instructor to clarify.)

.

5. Output signal levels

Estabish a high level on Pin 0 and a low level on Pin 1. Download the program and measure the voltages on pin 0 and 1. Also check to see if any changes have occurred to the other pins.

```
' High on 0, Low on 1
```

```
' {$STAMP BS2}
' {$PBASIC 2.5}

HIGH 0
LOW 1

END
```

6. Connecting LEDs

If you want to control a little light from the micro-controller, you can hook up a light emitting diode (LED). These little devices draw less current than an incandescent bulb and they come in different colours, although the original red is still the most common. (Be careful, sometimes yellow- or other-coloured LEDs actually are red when they light.)

The figure shows two ways to connect an LED. Both have a current-limiting resistor. This resistor is very important because without it the current limit of the I/O pin might be exceeded causing damage to the chip. The larger the current-limiting resistor, the less current is drawn from the output pin, and the dimmer is the LED when lit. Connect an LED to each of P0 and P1 as shown.

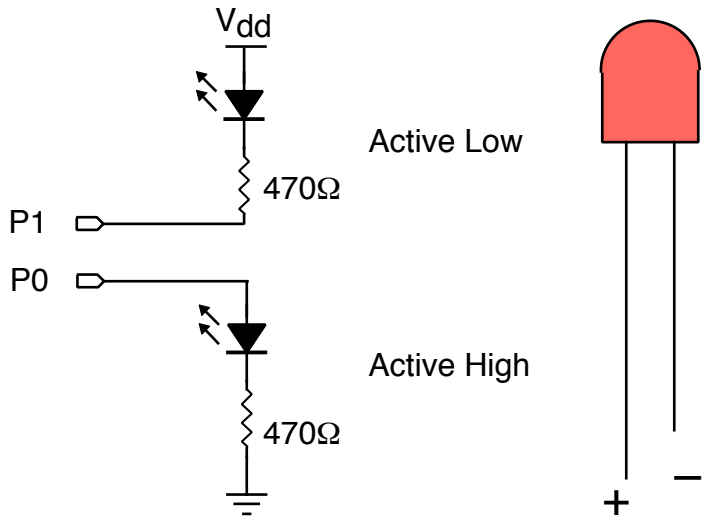

Fig 3: Two ways to connect an LED. The polarity of the LED is indicated by the leads: the longer one is the anode (+). When you bend the leads, leave the – one straight so you can tell which is longer.

After connecting the circuit, double check and it then run this program

```
' Turn on and off an active high led on Pin 0
' while turning off and on an active low led on Pin 1
' {$STAMP BS2}
' {$PBASIC 2.5}

DO
      HIGH 0
      HIGH 1
      PAUSE 500
      LOW 0
      LOW 1
      Pause 500
LOOP
END
```

Change the program to blink the lights faster and turn both lights on and off at the same time.

According to the specifications for the PIC16C57 each output pin can source 20 mA and can sink 25 mA. That means that the active low configuration is somewhat preferable because the current limit is higher. If one is connecting an array of LEDs the difference is even more important because each 4-bit port (P0-P3=Port A, P4-P7=Port B, etc.) has a 40 mA limit for sourcing and a 50 mA limit for sinking current.

If you wish to exceed these current limits in order to achieve a brighter light, you have to insert a pass transistor as shown in the following diagram.

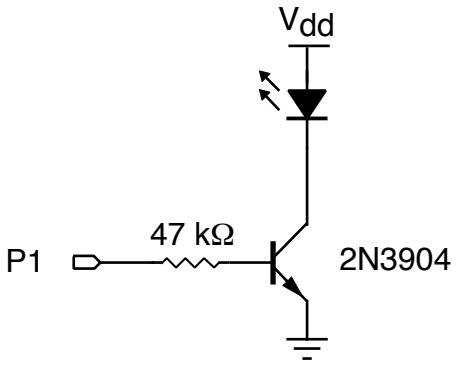

Fig. 4: Using a pass transistor and an NPN transistor one can greatly reduce the current drawn from an output pin. If the DC beta (h_{FE}) of this transistor is 100, then this circuit gives the same LED current but draws only 1% of the P1 current of a directly-connected LED.

When analysing such a circuit be aware than LEDs have a larger voltage drop than normal diodes. Red LEDs usually drop about 1.5 volts but other colours have even more. Luckily most are less than 5V.

Question: If 50 mA flows through the LED and its voltage drop is 1.5 V from Vdd then what is the power dissipation of the transistor?

7. Connecting a push-button switch on an input

The circuit in Fig. 5 shows a good way to connect a push-button switch to input one bit into the micro-controller. The switch has four terminals so that it will sit stably in a waffle or printed circuit board. A show terminals 1 and 4 are connected to each other as are terminals 2 and 3. If you are in doubt about which pins are connected, use your ohmmeter.

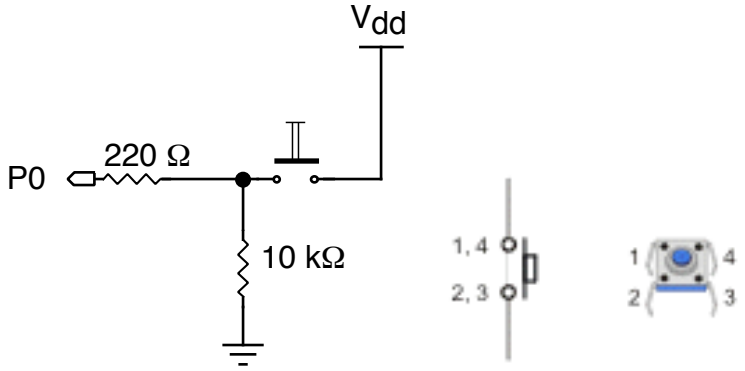

Fig. 5: A “Normally Low” input. Connecting a push button switch in this way insures that the input is never indeterminate. The small push button is designed to fit into an IC socket. It has four terminals but only two poles as shown.

The 220 Ω resistor to P0 is for safety, It is not strictly necessary, but if the pin is accidentally addressed as

a low-level output pin when the button is pressed then the resistor limits current damage. (Note that the Parallax Homework Board has 220 Ω resistors installed on all I/O pins. However, the Board of Education and other boards do not have such resistors.) The 10 kΩ pull-down resistor is fine for the CMOS micro-controller input; however, TTL inputs need a slightly smaller pull-down resistor such as 2.2kΩ. [I note that these little switches don't really seat too reliably in the waffle board. When new they seem to stick better, but there is a tendency to pop out.]

Use the switch to control the led with the following program.

```
' Control an LED with a push-button switch
' {$STAMP BS2}
' {$PBASIC 2.5}

DO
      IF (IN0) THEN
            HIGH 1
      ELSE
            LOW 1
      ENDIF
LOOP
END
```

8. Inputting a four-bit nibble using a dip switch.

The next circuit shows how to connect a 4-bit dip switch so that one can enter a hexadecimal number into an input port. One input port consists of four consecutive pins. For example, P4, P5, P6 and P7 constitute Port B and the entire nibble can be input by referencing variable INB.

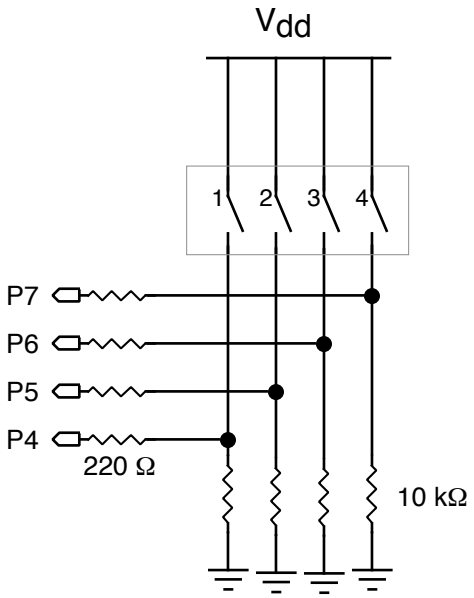

Fig 7: Connections of a four-bit dip switch to port B.

To print the value in hexadecimal on the Debug screen preface the value with HEX.

```
' Input a hex digit and print it on Debug screen
' {$STAMP BS2}
```

```
' {$PBASIC 2.5}

DEBUG CLS    ' Clear the screen of detritus
DO
 DEBUG HEX INB, CR       ' Print input nibble B in hexadecimal and go to new
line.
LOOP
END
```

9. Controlling Data Input with Handshaking

With the 4-bit nibble program, the computer reads the data every time the program goes through the loop. Sometimes you would like to indicate when data are to be read by the computer and when not to read the data. For this purpose one introduces another signal which one could call "Data Available". The data available signal is read (polled) by the computer every time it goes through its read loop and the 4-bit nibble is only read if that signal is active. After reading the data, the computer must then clear the data available so that it won't read the old data again. The following circuit illustrates the process. The push button sets the flip flop which puts a 1 in the Q output. The Q output of the D flip flop provides the DAV (data available) signal to pin 0. A 0 output on pin 1 is used to clear the flip flop through its reset pin.

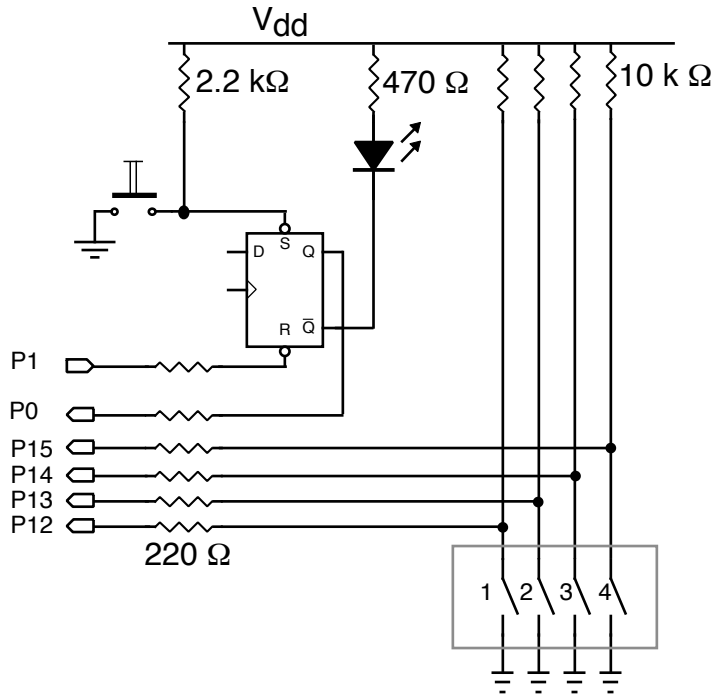

Fig 8: This circuit shows a D flip-flop being used as a Set-Reset flip flop which holds the "Data Available" (DAV) signal. The problem with this circuit is that if the button is still down while the data are read, then the DAV gets reset again.

```
' Input data bits with handshaking
' {$STAMP BS2}
' {$PBASIC 2.5}

Q var IN0
RESET var OUT1
i var word

INPUT 0
OUTPUT 1
```

```
DEBUG CLS    ' Clear the screen of detritus
i=1
RESET=1
 DEBUG bin inC.lowbit, CR      ' Print input bits and go to new line.
DO
IF (Q) THEN         ' Check if DAV is high
      i=i+1
 DEBUG dec i," ",Hex IND, CR
      RESET=0      'Reset the DAV flip flop
      RESET=1
 ENDIF
LOOP
END
```

If you run this program you'll notice a bug. The program continues to read data as long as the button is depressed, which is not what was intended. You want the program to read the data once per button press and the wait for it to be pressed again before reading another value. How would you solve this problem?

One solution is to pause for about 0.5 s after reading the data to give the user time to let go of the button. If the user keeps the button down for more than 0.5 s then you get a sort of "autorepeat" feature, which wasn't intended, but may be useful depending on the circumstances. Try inserting the pause where appropriate and see what happens. Obviously this solution may always be what you need.

The following circuit solves the problem by using the flip-flop in a slightly different way. The button is connected to the clock so than when the button is pressed and then released the positive edge of the pulse causes the 1 to be input from D. and sets the flip flop. Immediately after reading the data the computer resets the flip-flop and it doesn't get set again until the button is pressed and released again. At least, that's the theory. Take that 0.5 s pause out of the circuit and try it again with the modified circuit. What happens?

Draw a timing diagram of what is supposed to happen and then try to explain the actual behaviour.

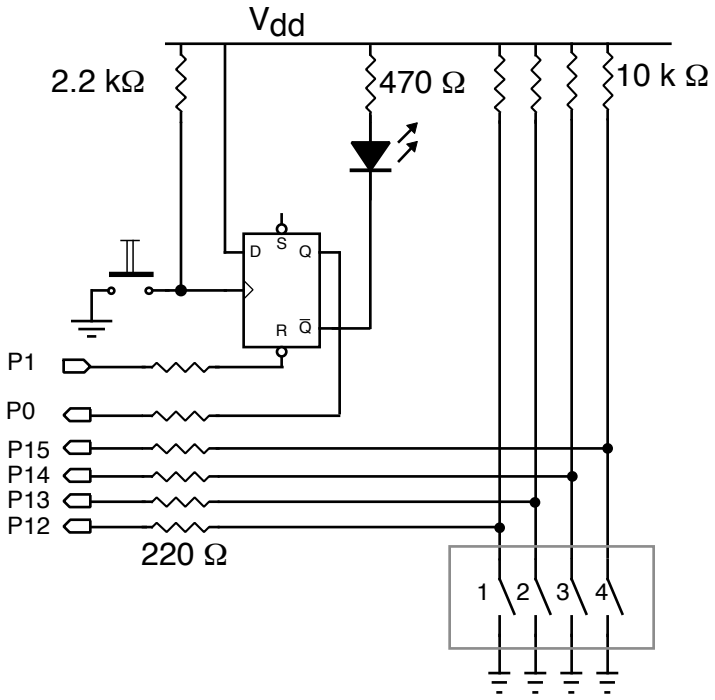

Fig 9: This circuit ensures that DAV is set only when the button is released so DAV is not inadvertently reset by keeping the button down too long. Switch bounce though does sometimes cause it to double read.

Sometimes this works as wished. You get one data value for each time you pressed the button and release it. The data is read upon release, which may be a problem, but not too bad. The real issue is that sometimes you get data when the button is pressed AND when it is released. Not good. Do you know what's happening?

Switch bounce...

Mechanical switches are often plagued by "switch bounce". After mechanical contacts touch, they may bounce open momentarily before settling back together again. This can cause multiple short pulses where only one is intended.

If you connect the scope to the button output, which goes to the CLK input, then you can probably catch the spurious switch bounce signal by triggering the scope in single sweep mode. It doesn't happen every time but if you press the button 10 or 20 times you should be able to see a spurious glitche just (20 to 100μs) before some of the real upward transitions.

The solution to switch bounce is to insert another set-reset flip-flop after the switch and use a DPDT switch as shown. Now the output of the first flip-flop will be a clean signal which won't cause spurious clocking of the second flip-flop.

If we have such a switch you can use the second D-flip-flop on the 74LS74 chip to debounce it. If you don't have it in your kit then just be aware of how to get a debounced signal and call it a day.

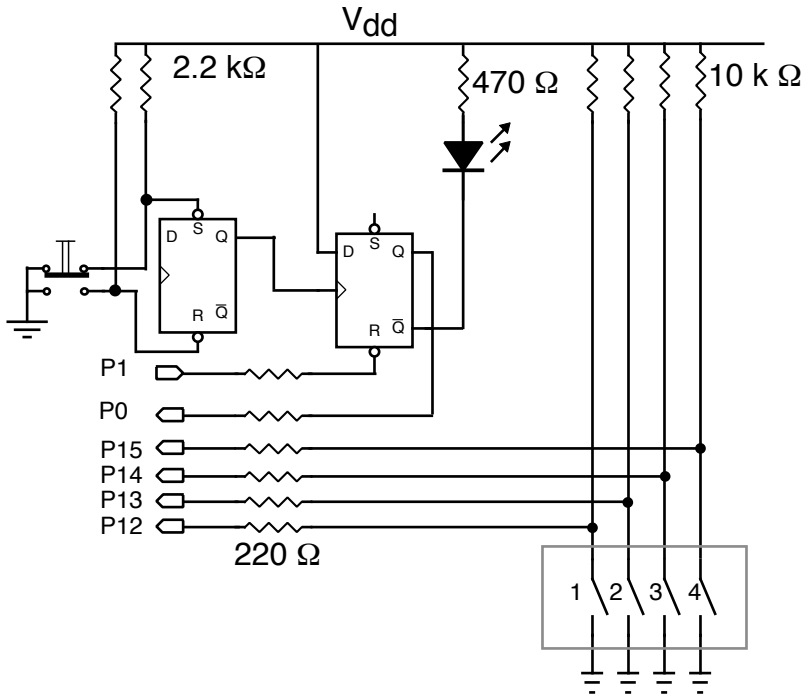

Fig 10: This circuit debounces the switch, but requires a DPDT (OR SPDT) switch.

10. Serial Data Input

The previous examples used parallel data input in which all data bits were transferred simultaneously to the micro-controller's port. Thus an eight-bit number would need to use eight of the sixteen I/O lines. Serial data transfer requires only one input port no matter how many bits are in the data. In addition one

clock signal must be provided in order to synchronize the process.

The following circuit uses the same basic pushbutton connection for both the clock and data signals. This and the associated program avoid problems encountered with the flip-flop handshaking, but is not strictly a handshaking protocol because the computer must be ready to read the data while the clock button is depressed.

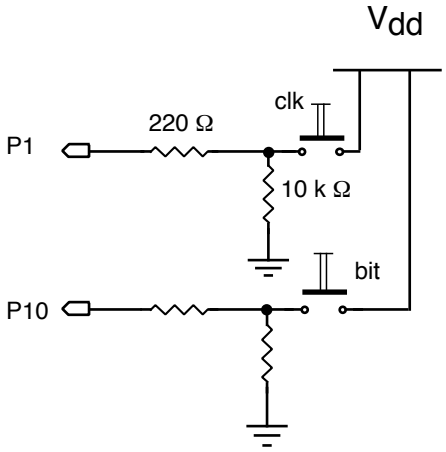

Fig 11: Circuit for serial data input using a clock and single data signal.

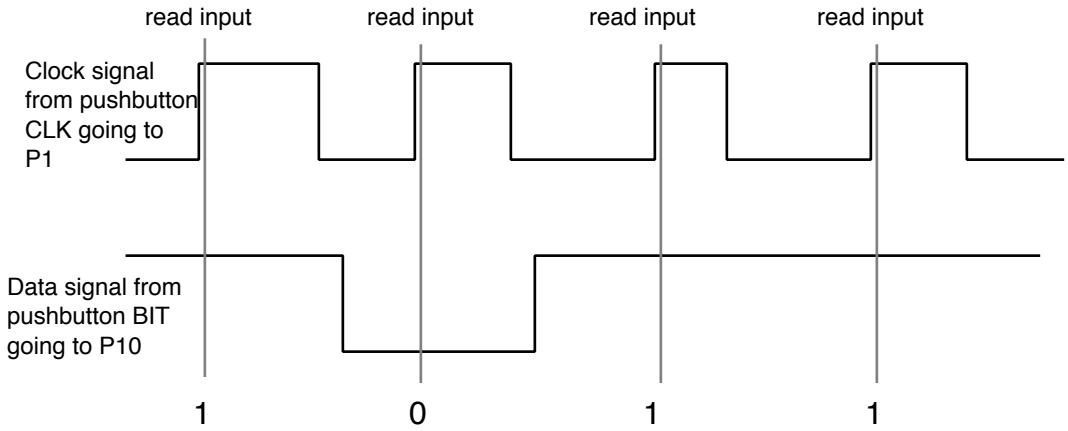

Fig 12: Timing diagram for serial input. This transfer is called asynchronous because the clock signal may be irregular.

The exercises in this lab can form the bases for many projects involving purely digital input and output. You should try one of the following for yourself as a mini-project. Keep these in mind and think of other applications which may be developed into a final project for the class.

Mini Projects: (optional, springboard for your class project)

1. Clever Hans: Read a hex digit, flash an LED that number of times.

2. Traffic lights with Crosswalk button

3. Logic gate tester. Test various 74'xxx series ic's for proper functioning.

4. Reaction timer: See WAMC Ch 3. This involves simple timing,

Printed with a Demo of Nisus Writer Express