Shadow Registers

Making Microcontroller I/O work correctly.

© Copyright 2011 John Main
# Table of Contents

Rights.................................................................................................................................3
Legal..................................................................................................................................3
Resources............................................................................................................................4
  PIC C Course..................................................................................................................4
  PIC Interrupt Secrets......................................................................................................4
  State Machine Secrets....................................................................................................4
The Problem With Some PIC IO ports...............................................................................5
  Why is there an I/O port problem?..................................................................................5
    Resistive problem..........................................................................................................5
    Capacitive problem......................................................................................................6
Why resistance or capacitance loading is a problem ......................................................6
Read Modify Write.............................................................................................................6
PORT hardware problem...................................................................................................7
18F Port Hardware Solution..............................................................................................8
Shadow Register Solution for other devices e.g. 16F.......................................................9
Rights

NOTICE:

You DO Have the Right Give Away This Report.

As long as you don’t change anything, use SPAM or generally do something you’re not supposed to do.

© Copyright 2011 John Main

http://www.best-microcontroller-projects.com

All rights reserved. This ebook may not be sold.

Legal

DISCLAIMER AND/OR LEGAL NOTICES:

Terms of use:

All software, hardware and information found in this ebook or associated with it shall be hereafter referred to as the product. All persons using or acting on the product shall be hereafter referred to as the user.

By using the product the user agrees to abide by the following conditions.

Disclaimer:

All products are provided 'as is' and without warranty of any kind, either expressed or implied. The entire risk as to the quality and performance of the product is borne by the user. Should the product prove defective in any respect, the user will assume the entire cost of any service and repair and will assume full liability for any indirect, special, incidental or consequential damages arising out of the use of or inability to use the product, including, without limitation, damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses, even if advised of the possibility thereof, and regardless of the legal or equitable theory (contract, tort or otherwise) upon which the claim is based.
Resources

PIC C Course

A Complete C course for PIC micros in 11 parts.

CLICK the image.

Or Click Here

PIC Interrupt Secrets

How to Use PIC Interrupts to Maximum effect.

CLICK the image.

Or Click Here.

State Machine Secrets.

How to use state machines to create Solutions to complex problems including fully debugged examples – one is coding a scrolling menu on a standard LCD display.

CLICK the Image

Or Click Here.
The Problem With Some PIC IO ports

Sometimes I/O does not operate the way you expect.

You may write a 1 to a single bit in a I/O port then read it back but you don't get a '1' returned you get a '0'.

You are probably expecting to read the same value back just as if you were writing to an internal register – writing a '1' you expect a '1' back but the hardware in an I/O ports is sometimes constructed differently. So lets have a look at why the I/O port can present you with a problem.

Note: I'm going to show you the 16F series I/O (generic port) that has a problem and the newer 18F series port that does not – so that you get the idea of looking at the datasheet to find out the exact construction of your port (Many of the I/O ports are multiple use so you have to look carefully at their hardware construction to see if there will be a problem or not).

Why is there an I/O port problem?

The problem is that the I/O port unlike and internal register has to connect to the real world and in the 'real world' there are capacitive and resistive loads that are much higher than in the internal (register environment). These resistive and capacitive loads depend on the circuits that you place on the pins so they are unknown until you try them out.

Resistive problem

When you drive a load even such a simple thing as an LED the microcontroller pin has to drive current through the LED.

It's true that most of the PIC I/O pins are capable of driving about 25mA but it's not the whole story since the internal resistance of the pin is also present – as more current is drawn so the output voltage level falls.

If you then read the port instead of reading a logic '1' as expected you may read a logic zero!
Capacitive problem

When you drive an output from a logic high to a logic low – depending on how much capacitance is present the voltage level may be held high for a while (even a few 100pF is enough to cause a problem).

So if you immediately read the port you will read back a logic '1' and not a logic '0'.

Why resistance or capacitance loading is a problem

In both cases you probably want to read back the I/O port using it as a memory device i.e. if you turn on an LED you might read back the port to see the current state and then turn it off again i.e. flash the LED. If you don't read back the correct logic level then the program will hang always reading back a zero will keep the LED on all the time – not what you want.

Read Modify Write

Most PIC instructions perform extremely efficient actions by performing a so called Read-Modify-Write action but all in one instruction e.g.

SETB PORTB.0 ; A bad idea don't use this on PORTs. It is fine on registers.

Other R-M-W assembler instructions

BSF, BCF, ANDWF, IORWF, XORWF

If you use these with an I/O port then you run the risk of having errors because the port value at the pin itself is being read.
PORT hardware problem

The problem is the construction of the I/O port. The diagram below is for a generic 16F series I/O port.

![Block Diagram of RA0/AN0:RA1/AN1 Pins]

Here you can see the I/O pin connected to the driver and protection diodes. The problem is that the data is read back directly from the I/O pin as shown by the red arrow and the yellow highlight path.

If the I/O pin is loaded in any way then the read back data can be in error i.e. different to the value that is being driven to the port.
18F Port Hardware Solution

In the 18F series devices each port has an extra memory mapped data storage element called LAT (short for latch). Either the original PORTA or LATA can be used to read and write data from/to the port but the LAT register reports back the data that was written to the port not the actual value at the port itself.

This gets round the problem since you can use the LAT registers when you must only have the expected data values and not the 'capacitively or resistive loaded values and it also allows Read-Modify-Write instructions to perform correctly.

Note: You can still read the port value itself using the original PORTB register.
Shadow Register Solution for other devices e.g. 16F

For devices other than the 18F series the solution is to fake it!

You have to define a register that will always be associated with each specific port. This is called a shadow register since it always copies the exact state of the 'real' register.

In C you could write

```c
short PORTB_shdw = 0;
```

Whenever you want to write to PORTB you first update the shadow and then write it to the 'real' port;

```c
PORTB_shdw = 0x44;
PORTB = PORTB_shdw;
```

To read the port :

```c
port_val = PORTB_shdw;
```

To make the system transparent use macro functions (#define) in C to produce in line code.

*Note: all operations must be performed on the shadow register e.g. Bit manipulation and then the shadow register must be written to the port.*