Microcontrollers

Class 0: Overview and Setup

February 28, 2011

Outline

Outline

Outline

What is a microcontroller?

It's a whole computer on a chip:

- Write programs in various languages (C, assembly, BASIC)
- CPU (1-20MHz)
- Dynamic memory (SRAM)
- Non-volatile memory (Flash ROM and EEPROM)

But it's a *very* little computer:

- 8-bit words
- Not much memory (8kb program space, 512 bytes SRAM)
- No operating system
- Low-level input/output
- = halfway between a "component" and a "computer"

What can it do?

Damn-near anything!

- Super-fancy Blinkers
- Robots
- ROM readers
- Phone dialers
- Noisemakers
- GPS dataloggers
- What do you need to do?

Outline

Basic Functionality

What do they Actually Do?

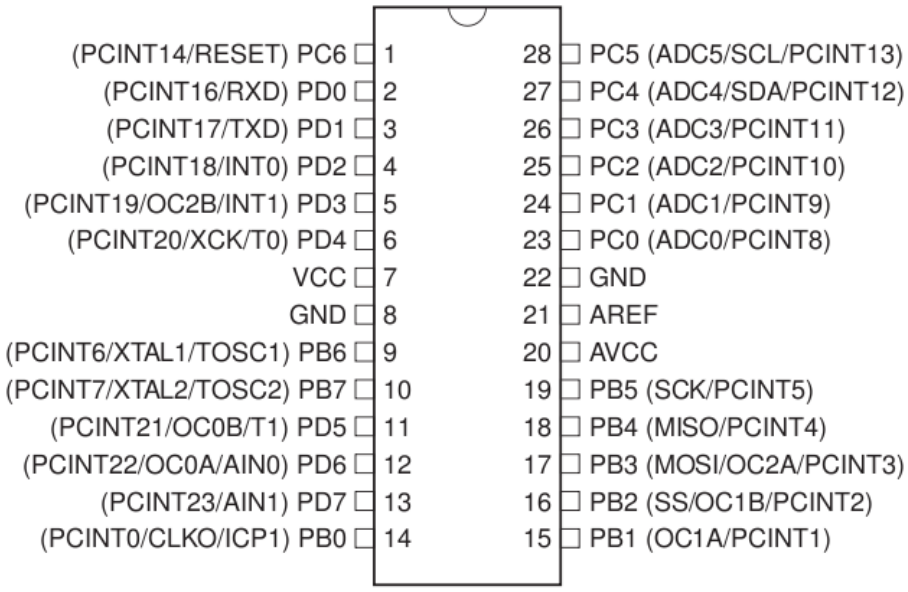
- Output: 5v or 0v for each pin.
  (Light up LEDs, flip switches, spin motors)
- Input: Digital (pushbuttons, threshold sensors) or
  Analog-to-Digital conversion (light levels, audio waveforms)
- Neither: ("Hi-Z") plays like it's disconnected from the circuit
- Pulse-width Modulation (PWM): Flip the digital output on and off quickly. Simple way of making an analog signal with a digital output pin

Other Stuff

Useful features

- Timers: Our chips have (3) internal clocks, useful for both timing and scheduling events
- (Timers also make doing PWM and audio stuff easy)
- Serial I/O: built-in hardware-level routines for USART, SPI, I2C serial protocols
- Interrupts: Allow you to call a subroutine whenever a button is pushed or a certain timer event occurs (and more)

Pinouts



The Basic Workflow

What will we actually be doing?

- Write code in C (using whatever you want)
- Cross-compile for the chip $\rightarrow$ the machine-code version of your code
- Transfer the code to the chip:
  – Programmer to talk to the chip
  – Software to run the programmer
- Get feedback/debug until it works

The "Toolchain"

How to get firmware into the chippy

- Cross-compiler: GNU GCC and a bunch of help from avr-libc
- AVRdude: knows how to run a bunch of programmers
- Programmer (USBtiny) or a bootloader already flashed into the chip
- Usually a Makefile to compile and flash for you in one step
- `http://wiki.hacdc.org/index.php/Installing_AVR_Toolchain`

Outline

Blinky LED Demo

You've gotta start somewhere...

- Wire up: Connect an LED from pin PB0 to a resistor to ground. See the example board.
- On the LED, long lead is positive and goes to PB0
  Short lead goes to a (120 Ω) resistor.
- Now open up the file `LED_Demo.c`

Our First Program

Structure

- `#include` are directives to load up code from other files
- `#define` sets macro variables that are substituted into values before the program is compiled
- Defining your pinouts for the chip is very nice because it makes readable code and documents how you want the thing wired up.
- Main function – this is what gets run when the chip wakes up
- `while(1)` endless loop runs forever

Bits, Registers, and Values

Configuring the Chip

- Memory locations ("registers") map directly to hardware, flipping internal switches on or off
- Each register byte is 8 switches, each bit a switch
- We can read/write/edit the numerical values in these registers to change the chip's configuration or state
- `DDRB = _BV(LED);`
- "Data direction register B" has 8 switches, set to 0 for input or 1 for output
- Here, we're setting up the LED'th pin of port B for output

Writing Output

```
PORTB = _BV(LED);
```

- Pins are grouped together in sets of eight into ports
- `PORTB` is another register, containing a switch for each of eight pins
- When the `DDR` is set for output, writing a 0 to a `PORT` register bit sets it to the ground voltage, and writing a 1 sets it to logic high.
- `_BV(i)` shifts a bit into the *i*'th position, read Bit Value
- So we're turning on the LED'th bit/switch/pin in Port B, lighting up our LED!
- Note the use of a `#define LED` macro to make the code readable

Taking Blinking to the Extreme

Getting Fancier

- For homework, wire up all eight LEDs to all of the PortB pins
- Now you can write a byte directly to `PORTB` and it'll be displayed on the blinkenlights
- What can we do with this? Fun patterns.

Homework

Cylon Eyes

- The cylons/knight-rider thing has been done to death on the intertubes. Here's your chance to find out why!
- Solder up 8 LEDs and resistors to complete the Port B outputs
- Using what you know about turning on different pins, make a cylon-eyes type scroller
- Can you make it look better? Can you think of cooler patterns?

Homework Extension

POV

- Make the delay in your cylon eyes very very quick
- Wave the thing around in the air
- Voila!
- Now code up cool patterns for it.
- If you know enough C, you can make good use of character arrays and for loops here.

The End

Outline