Intro to Microcontrollers

Class 2: Output: Bit Math, Cylon Eyes, and PWM

September 23, 2008

Outline

Outline

Hello World Example

Blinkies!

- Last class, showed an example that turned a pin on and off
- Sections of the C code:
  preamble/includes/defines,
  function definitions (didn't have any),
  main function (chip initialization and endless loop)
- The main loop twiddled a bit back and forth in a memory register, and that made Vcc and GND volts appear on a particular pin.
- But let's flesh that all out a little more...

Registers

Special memory locations

- Usually we think of memory as being a place to store info
- In micros, some special memory regions change the way the chip behaves: Registers
- DDRx register from initialization stage of the demo blinky program
- Writing a "one" to a bit in the DDRx register sets up a corresponding pin for output
- There's a similar mapping from the PORTx register to the output of the pins: writing a "one" to a bit in PORTx sets the corresponding pin at Vcc, "zero" to GND.
- These two registers (per port) control initialization of the pins as input/output and allow you to set the voltage on the pins

Addressing the Pins

Writing bits to registers

- So, say we're working on PORTD, and we want to set pin PD2 to 5v (to light up a LED)
- Write a 1 in the 2 place
- Want something like: PORTD = 00000100
- Unfortunately, no direct binary access, but we can figure out the value in decimal or hexadecimal
- 00000100 = 4 or 0x0004
- So `PORTD = 4;` will do it

Addressing the Pins II

Writing two bits to registers

- Now say we want PD2 and PD3 both on
- PORTD = 00001100
- `PORTD = 12;` or `PORTD = 0x000b;` will work
- (Quick, what's 11001101?)
- Not readable or easy to manipulate; hard to calculate
- Last class, I used the `_BV()` macro
- This class, we'll build up to it

Outline

The Math

Bit-shift Operators

- <<: Left shift
- >>: Right shift

Binary logic

- `&` : AND
- `|`: OR
- `^`: XOR
- `~`: NOT

Bit-shifting

Left shift: <<

- Very handy: say you want a 1 in the pin-3 place: 00001000
- Start with 1: 00000001
- Shift it over 3:
  `1 << 3` = 00001000
- Or using the pin-name macros: `1 << PD3`
- `#define _BV(bit) (1 << (bit))`

Right shift: >>

- Start with 12: 00001100
- Shift 2:
  `12 >> 2` = 00000011
- Note that right-shift is like dividing by 2^n: handy
- (Similarly, left-shift is like multiplying by 2^n)

Using Shifts

Quiz:

- Start with 10: 00001010
- What's `(10 >> 1)`? 00000101 ($10/2 = 5$)
- What's `(10 >> 2)`? 00000010 ($10/4 = 2$)
- What's `(10 << 4)`? 10100000 ($10 * 16 = 160$)
- What's `(10 << 5)`? 01000000 ($10 * 32 = 320 = 64$)

Practical examples:

- `PORTD = (1 << 3);`
- `PORTD = (1 << PD3);`
- `PORTD = (1 << (1+2));`
- `j = 3; PORTD = (1 << j);`
- `j = 3; PORTD = _BV(j);`

Set Two Pins

Addition:

- Say we want PD3 and PD4 both on
- Add them together?
- `PORTD = _BV(PD3) + _BV(PD4);` will work
- After all:

$$\begin{array}{rr} & 00001000 \\ + & 00010000 \\ = & 00011000 \end{array}$$

- Works if you're just setting the port using `PORT = something;`

Turning Pins On: Part II

What's better than adding?

- What if you don't know (or care) what LEDs are already on, but you want to turn on PD3?
- `PORTD = PORTD + _BV(PD3);`?
- If PD3 is already on:

```
  00000100
+ 00000100   Ouch!
= 00001000
```

- Could be worse:

```
  01111100
+ 00000100
= 10000000
```

- We need an OR statement

Turning on Additional Pins

The OR statement: |

- OR turns on a bit if this bit is on or if that bit is on
- `PORTD = PORTD | _BV(PD3);`
- If PD3 is not on:

```
  00000000
| 00000100
= 00000100
```

- If PD3 is already on:

```
  00000100
| 00000100   Yay!
= 00000100
```

- Turn on PD3, PD4, PD5?
  `PORTD = PORTD | (_BV(PD3) + _BV(PD4) + _BV(PD5));`
- And here's a nice shorthand: $X = X + Y \quad \rightarrow \quad X \mathrel{+}= Y$
  `PORTD |= (_BV(PD3) + _BV(PD4) + _BV(PD5));`

Turning Pins Off

The AND statement: &

- AND turns on a bit if this bit and that bit are *both* on
  11111110
- & 10001101
  = 10001100
- Can think of AND as masking out the the off bits
- Use it to turn off PD3: `PORTD &= 11110111`?
- `PORTD &= (_BV(PD7) + _BV(PD6) + ... skip PD3 ... + _BV(PD0) );`
- Works, but it sucks.

Turning Pins Off II

The NOT statement: ~

- There must be a better way to make 11110111
- `_BV(PD3)` = 00001000
- `~_BV(PD3)` = 11110111
- Turn off PD3: `PORTD &= ~_BV(PD3);`
- Turn off PD3 and PD4:
  `PORTD &= ~(_BV(PD3) + _BV(PD4));`
  Careful with those parentheses!
- Also remember this in terms of the fundamentals:
  `PORTD &= ~((1 << PD3) + (1 << PD4));`
  `PORTD &= ~((1 << 3) + (1 << 4));`

Toggling a Pin

The XOR statement: ^

- We won't use it today, but it's handy to be able to toggle a bit
- `PORTD ^= _BV(PD3);`
- ```
    11111001
  ^ 00001000
  = 11110001

    11110001
  ^ 00001000
  = 11111001
  ```

One Last Part...

...then Cylon Eyes

- So we know how to turn on bits, and how to turn them off
- How do we make cylon eyes?
- Start with light 0 on.
  Turn off the 0th, turn on the 1st, pause
  turn off the 1st, turn on the 2nd, pause
  etc
- `PORTD &= ~_BV(PD0); PORTD |= _BV(PD1); delay`
  `PORTD &= ~_BV(PD1); PORTD |= _BV(PD2); delay` etc.
- {`PORTD &= ~_BV(i) ; PORTD |= _BV(i+1); delay` }
- And make `i` range from 0 to 7 and back again
  (being very careful about endpoints)
  ```

Basic Looping

The For loop

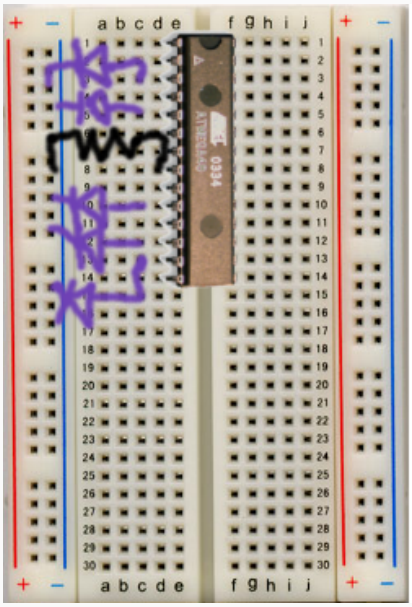
- `for(i=0; i < 7; i = i + 1){...}`
- Repeats the block in parentheses a bunch of times.
- First time, `i` $= 0$.
- Then it checks if `i` < 7.
  If not, it skips the block and moves on.
  If so, it executes the next command and then the block.
- So in our case, it executes the block with `i`$= 0, 1, 2, 3, 4, 5, 6$ and then is done.
- `for(i=7; i > 0; i = i - 1){...}` and a different block will bring it back down
- `i`$= 7, 6, 5, 4, 3, 2, 1$

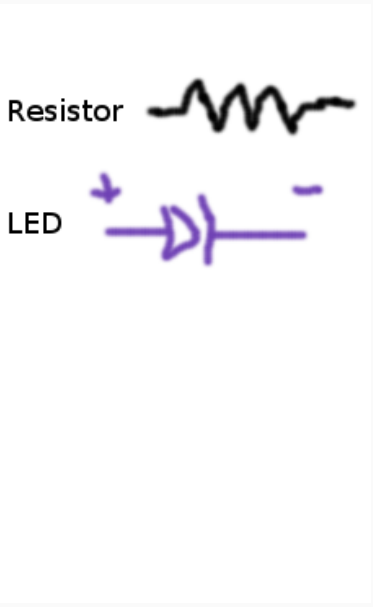
Cylon Eyes Setup

In words:

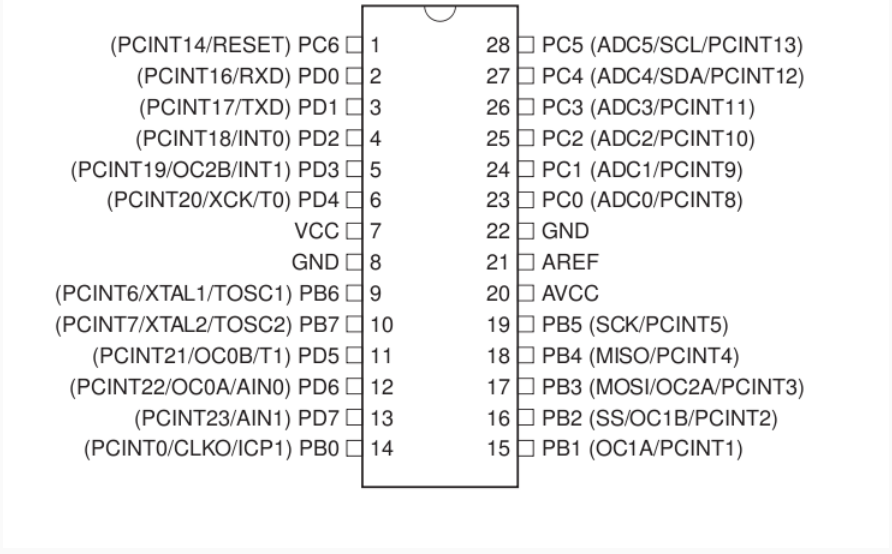
- We want 8 LEDs plugged in, one per pin in PORTD
- Ideally, we'd use a current-limiting resistor per LED
- If you haven't soldered up blinkenlights, there's a nice cheat: connect one resistor from the ground pin to the ground bus then each LED from AVR pin to the "ground" bus

Cylon Eyes Setup II

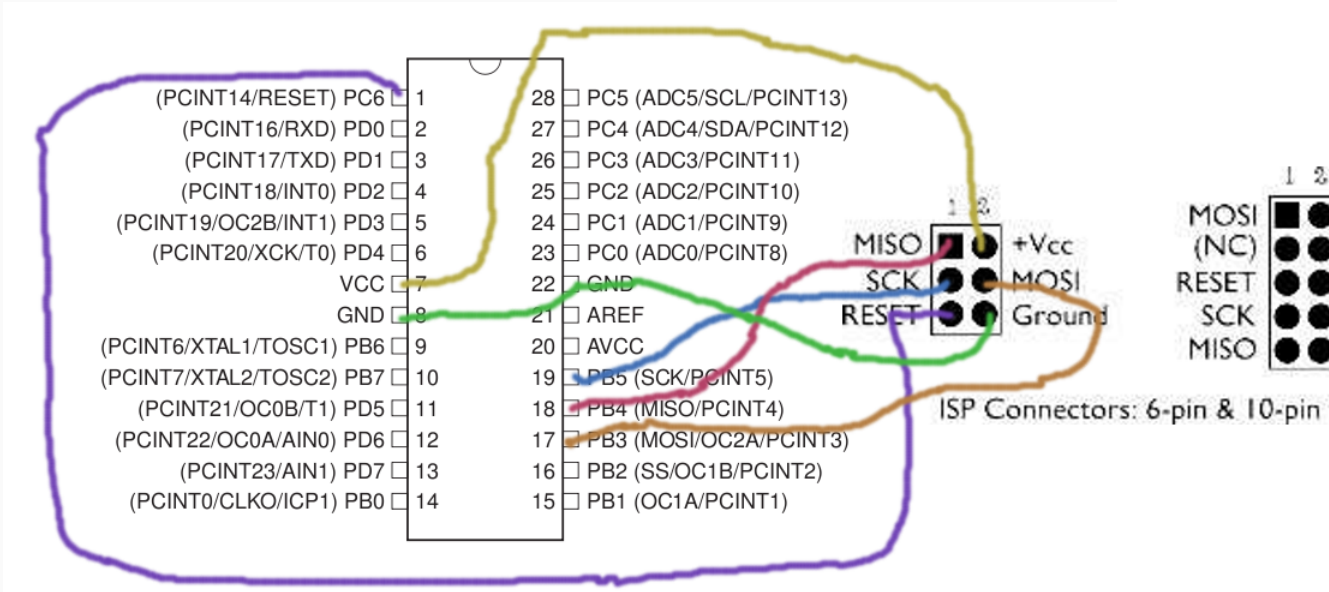




AVR Mega 48 Pinouts



Programming Hookup



Outline

Pulse-Width Modulation

Making analog from digital

- So we can turn lights on and off...what about dimming?
- The trick is to turn them off so quickly the eye can't see
- How? Easy, actually. Can do it just with `for` loops.
- Make a big loop. Turn on the light at the beginning of the loop, turn it off somewhere in the middle.

PWM Code

Two Examples

- introPWM.c is basic: just keeps the LED at a given brightness
- fadingPWM.c is more fun: fades the LED in and out
- Homework: Crossfading cylon eyes
- Hint: can use same outer loop to fade one light in while the other fades out

The End

Outline