What is AVR ?

- Modified Harvard architecture 8-bit RISC single chip microcontroller
- Complete System-on-a-chip
  - On Board Memory (FLASH, SRAM & EEPROM)
  - On Board Peripherals
- Advanced (for 8 bit processors) technology
- Developed by Atmel in 1996
- First In-house CPU design by Atmel

AVR Family

- 8 Bit tinyAVR
  - Small package – as small as 6 pins
- 8 Bit megaAVR
  - Wide variety of configurations and packages
- 8 / 16 Bit AVR XMEGA
  - Second Generation Technology
- 32 Bit AVR UC3
  - Higher computational throughput

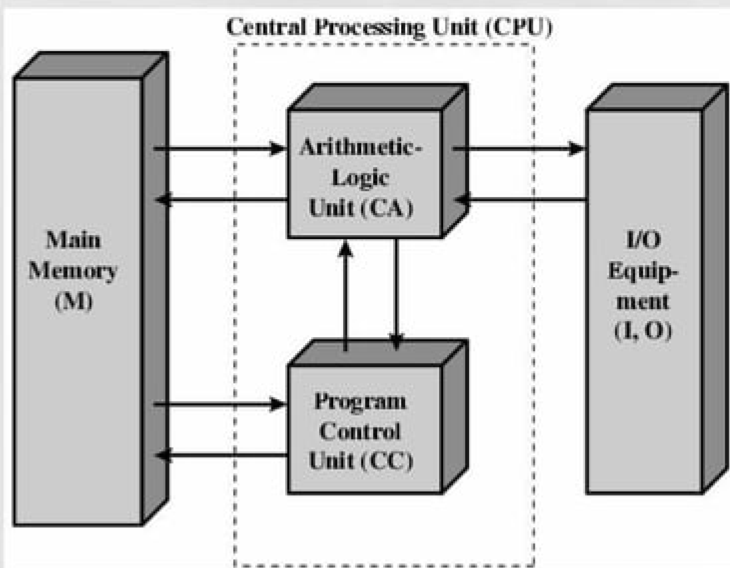
Presentation Overview

- Processor core
- Peripherals
- Hardware Example – Polulu 3pi robot
- Development Environments
- Software Example - PID algorithm walkthrough

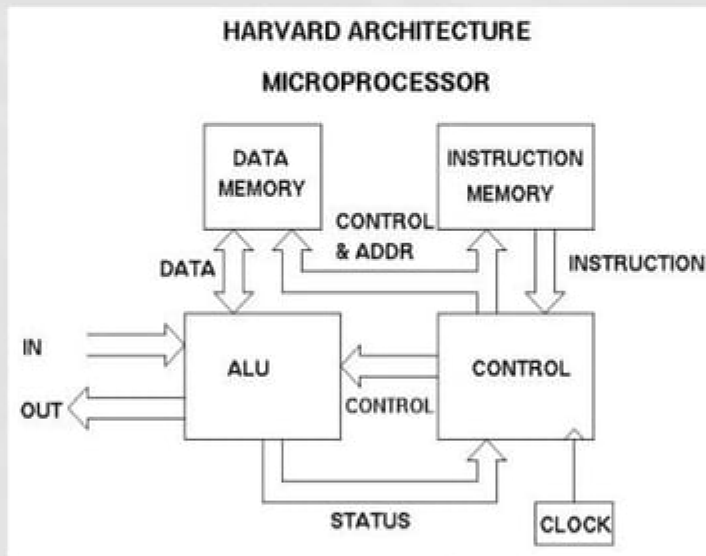
Processor Core

- What is Harvard Architecture?
- Before we can answer that…

Von Neumann Model for Stored Program Computers



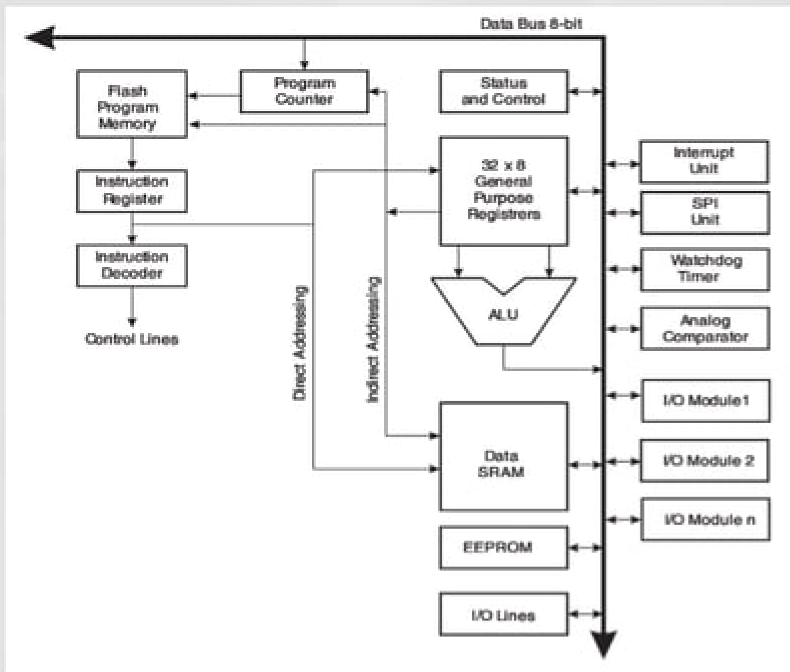
Harvard Architecture



Harvard Architecture Advantages

- Separate instruction and data paths
- Simultaneous accesses to instructions & data
- Hardware can be optimized for access type and bus width.

AVR Architecture



Modified Harvard Architecture

- Special instructions can access data from program space.
- Data memory is more expensive than program memory
- Don’t waste data memory for non-volatile data

What is RISC?

- Reduced Instruction Set Computer
- As compared to Complex Instruction Set Computers, i.e. x86
- Assumption: Simpler instructions execute faster
- Optimized most used instructions
- Other RISC machines: ARM, PowerPC, SPARC
- Became popular in mid 1990s

Characteristics of RISC Processors

- Faster clock rates
- Single cycle instructions (20 MIPS @ 20 MHz)
- Better compiler optimization
- Typically no divide instruction in core

AVR Register File

- 32 8 Bit registers
- Mapped to address 0-31 in data space
- Most instructions can access any register and complete in one cycle
- Last 3 register pairs can be used as 3 16 bit index registers
- 32 bit stack pointer

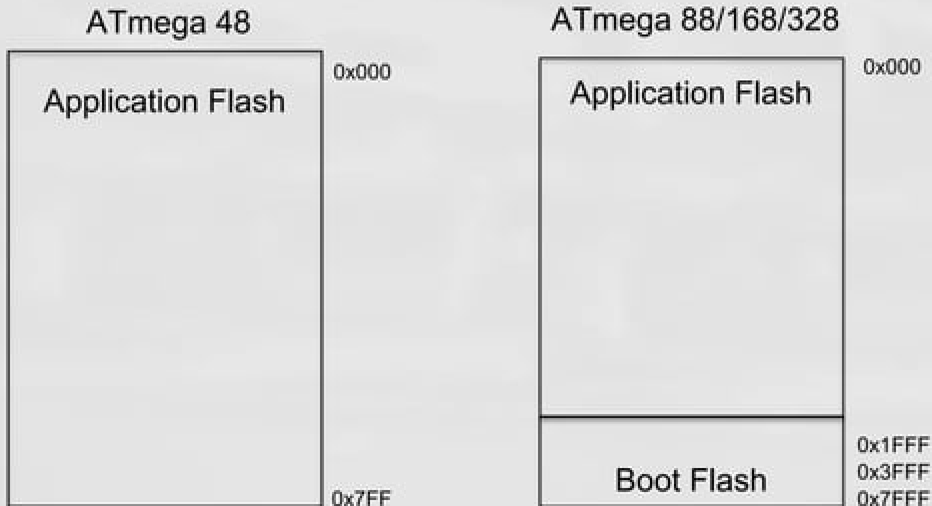
Register File

AVR Memory
FLASH

- Non-volatile program space storage
- 16 Bit width
- Some devices have separate lockable boot section
- At least 10,000 write/erase cycles

AVR Memories

FLASH – Memory Map



AVR Memories

SRAM

- Data space storage
- 8 Bit width

AVR Memories

SRAM - Memory Map

Region	Address
32 Registers	0x0000 – 0x001F
64 I/O Registers	0x0020 – 0x005F
160 External I/O Reg	0x00060– 0x00FF
Internal SRAM (512/1024/2048x8)	0x0100 – 0x04FF/0x6FF/0x8FF
External SRAM	

AVR Memories
EEPROM

- Electrically Erasable Programmable Read Only Memory
- 8 bit width
- Requires special write sequence
- Non-volatile storage for program specific data, constants, etc.
- At least 100,000 write/erase cycles

AVR Memories

DEVICE	FLASH	EEPROM	SRAM
ATmega48A	4K Bytes	256 Bytes	512 Bytes
ATmega48PA	4K Bytes	256 Bytes	512 Bytes
ATmega88A	8K Bytes	512 Bytes	1K Bytes
ATmega88PA	8K Bytes	512 Bytes	1K Bytes
ATmega168A	16K Bytes	512 Bytes	1K Bytes
ATmega168PA	16K Bytes	512 Bytes	1K Bytes
ATmega328	32K Bytes	1K Bytes	2K Bytes
ATmega328P	32K Bytes	1K Bytes	2K Bytes

Memory Mapped I/O Space

- I/O registers visible in data space
  - I/O can be accessed using same instructions as data
  - Compilers can treat I/O space as data access
- Bit manipulation instructions
  - Set/Clear single I/O bits
  - Only work on lower memory addresses

ALU – Arithmetic Logic Unit

- Directly connected to all 32 general purpose registers
- Operations between registers executed within a single clock cycle
- Supports arithmetic, logic and bit functions
- On-chip 2-cycle Multiplier

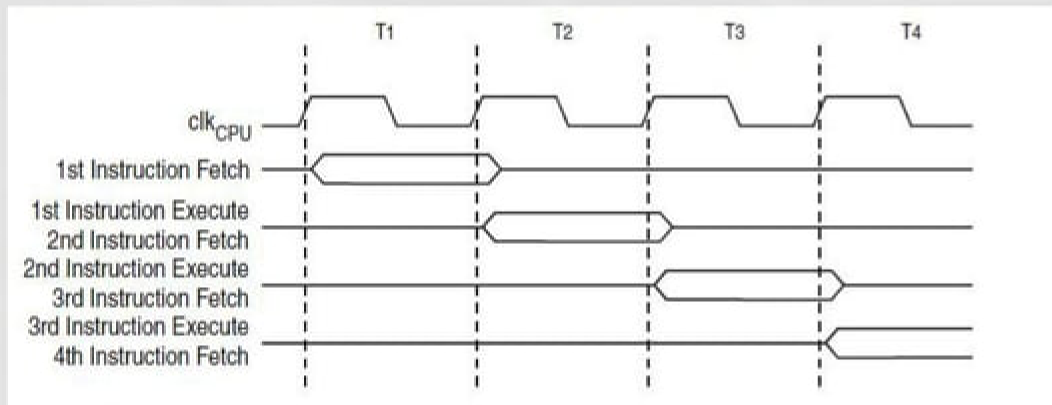
Instruction Set

- 131 instructions
  - Arithmetic & Logic
  - Branch
  - Bit set/clear/test
  - Data transfer
  - MCU control

Instruction Timing

- Register ↔ register in 1 cycle
- Register ↔ memory in 2 cycles
- Branch instruction 1-2 cycles
- Subroutine call & return 3-5 cycles
- Some operations may take longer for external memory

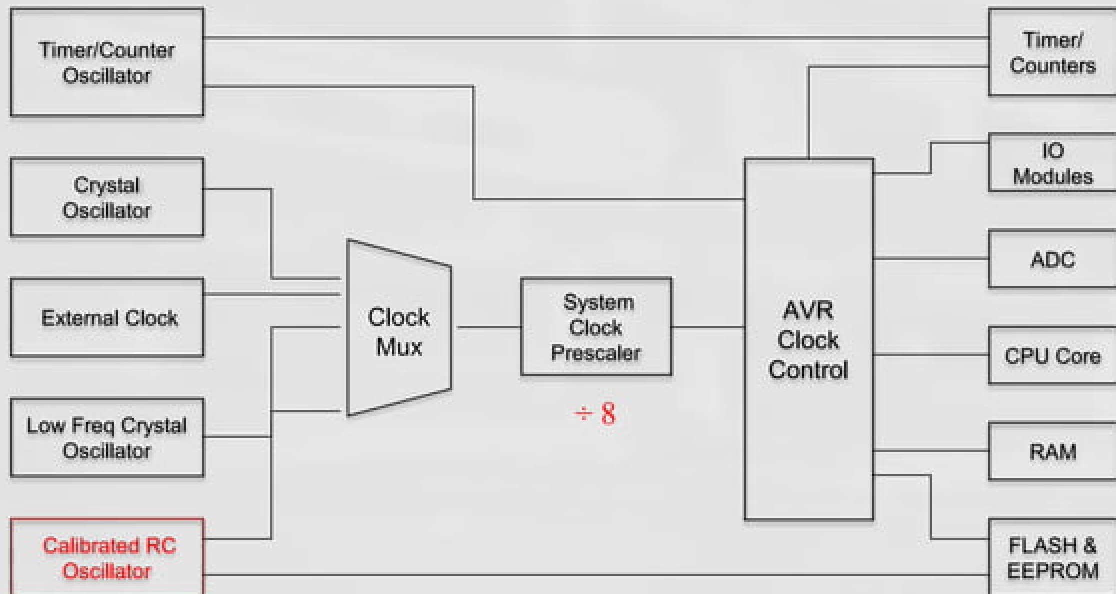
Pipelined Execution



AVR Clock System

- Clock control module generates clocks for memory and IO devices
- Multiple internal clock sources
- Provisions for external crystal clock source (max 20 MHz)
- Default is internal RC 8 MHz oscillator with ÷ 8 prescale yielding 1 MHz CPU clock
- Default is only 5-10% accurate

Clock Sources



Power Management

- Multiple power down modes
  - Power down mode
    - Wake on external reset or watchdog reset
  - Power save mode
    - Wake on timer events
  - Several standby modes
- Unused modules can be shut down

Reset Sources

- Power on reset
- External reset
- Watchdog system reset
- Brown out detect (BOD) reset

Interrupts

- ATmega328 has 26 reset/interrupt sources
- 1 Reset source
- 2 External interrupt sources
- I/O Pin state change on all 24 GPIO pins
- Peripheral device events

Interrupt Vectors

- Each vector is a 2 word jump instruction
- Vectors start at program memory address 0
- Reset vector is at address 0
- Sample vector table:

```
Address Labels        Code          Comments
0x0000                jmp RESET     ; Reset Handler
0x0002                jmp EXT_INT0  ; IRQ0 Handler
0x0004                jmp EXT_INT1  ; IRQ1 Handler
0x0006                jmp PCINT0    ; PCINT0 Handler
0x0008                jmp PCINT1    ; PCINT1 Handler
                      ...
```

Fuses

- Fuses configure system parameters
  - Clock selection and options
  - Boot options
  - Some IO pin configurations
  - Reset options
- Three 8 bit fuse registers
- Use caution! Some configurations can put the device in an unusable state!

ATmega Peripherals

- 23 General Purpose IO Bits
- Two 8 bit & one 16 bit timer/counters
- Real time counter with separate oscillator
- 6 PWM Channels
- 6 or 8 ADC channels (depends on package)
- Serial USART
- SPI & I^2C Serial Interfaces
- Analog comparator
- Programmable watchdog timer

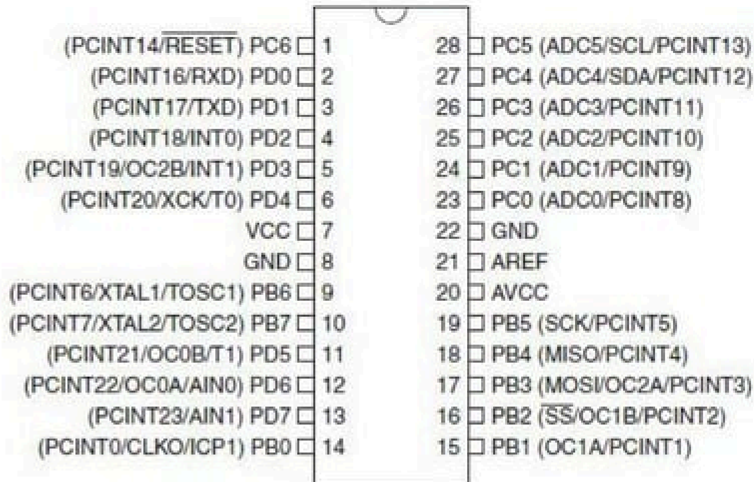
General Purpose IO Ports

- Three 8 Bit IO Ports
  - Port B, Port C & Port D
  - Pins identified as PBx, PCx or PDx (x=0..7)
- Each pin can be configured as:
  - Input with internal pull-up
  - Input with no pull-up
  - Output low
  - Output high

Alternate Port Functions

- Most port pins have alternate functions
- Internal peripherals use the alternate functions
- Each port pin can be assigned only one function at a time

Alternate Pins for PDIP Package



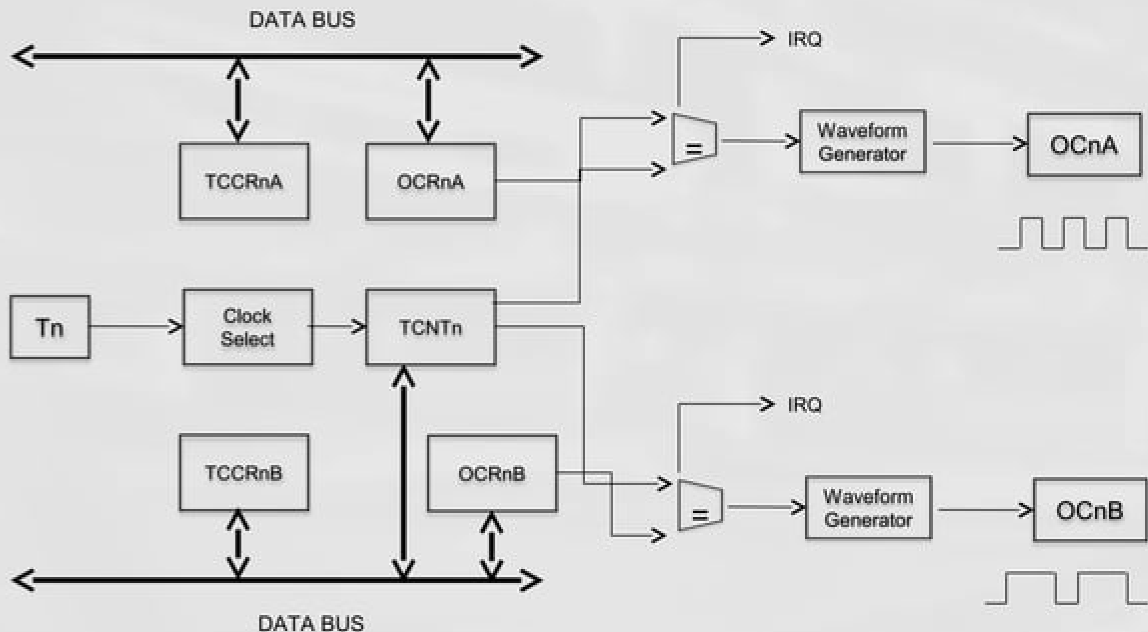
Timer / Counters

- 8/16 Bit register
  - Increments or decrements on every clock cycle
  - Can be read on data bus
  - Output feeds waveform generator
- Clock Sources
  - Internal from clock prescaler
  - External Tn Pin (Uses 1 port pin)

Timer / Counters

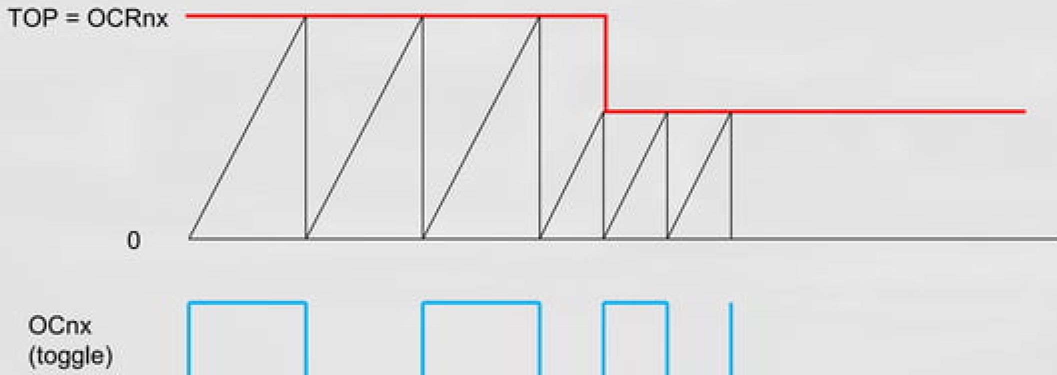
- Multiple Operating modes
  - Simple timer / counter
  - Output Compare Function
    - Waveform generator
      - Clear/set/toggle on match
    - Frequency control
    - Pulse Width Modulation (PWM)

Timer / Counter Block Diagram



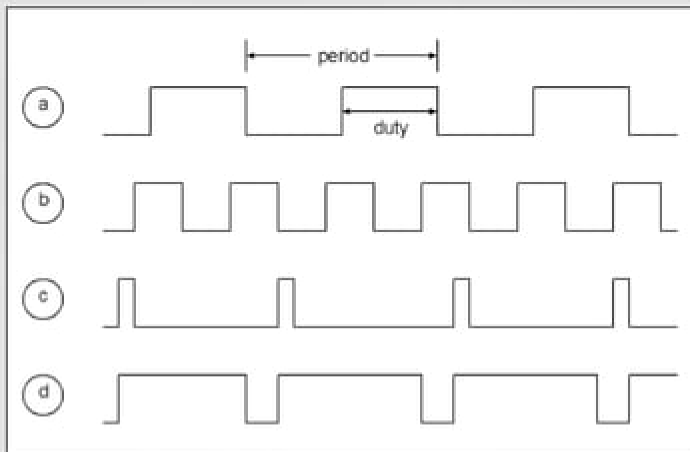
Controlling Frequency

- Use Clear Timer on Compare Match (CTC) Mode
- OCnx Toggles on Compare Match



Pulse Width Modulation (PWM)

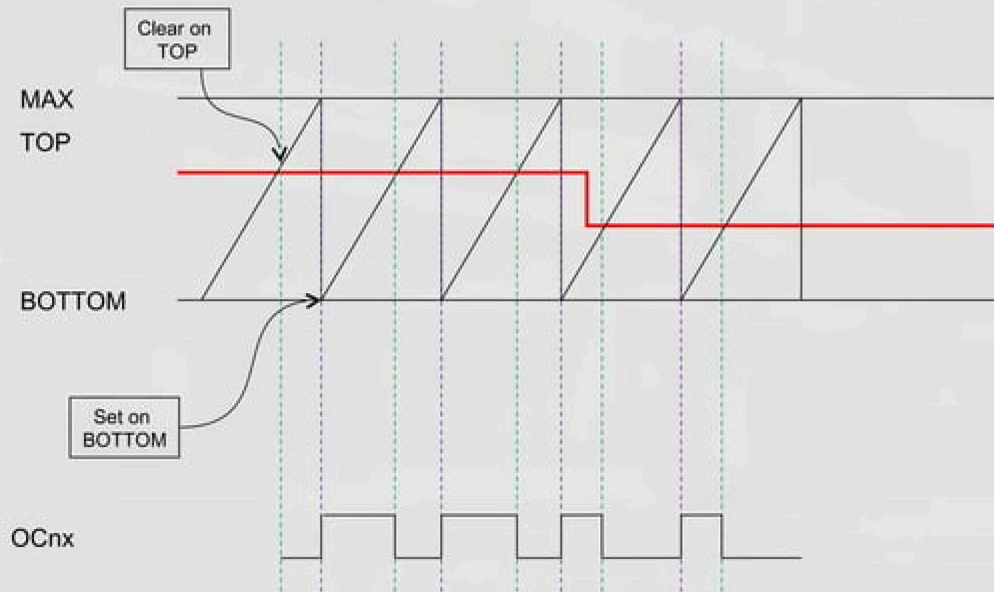
- Dynamically change duty cycle of a waveform
- Used to control motor speed



PWM with AVR Timers

- Fast PWM Mode
  - Counter counts from BOTTOM (0) to MAX
  - Counter reset to 0 at MAX
  - OCnx cleared at TOP
  - OCnx set at BOTTOM

PWM Waveform Generation



Timer Capture Mode

- Timer 1 has capture mode
- Capture can be triggered by ICP1 pin or ACO from analog comparator
- Capture event copies timer into input capture register ICR1
- Can be used to time external events or measure pulse widths
- Range finders generate pulse width proportional to distance

Analog to Digital Converter (ADC)

- 10 Bit Successive Approximation ADC
- 8 Channel multiplexer using port pins ADC0-7
- Max conversion time 260 μsec.

Analog Comparator

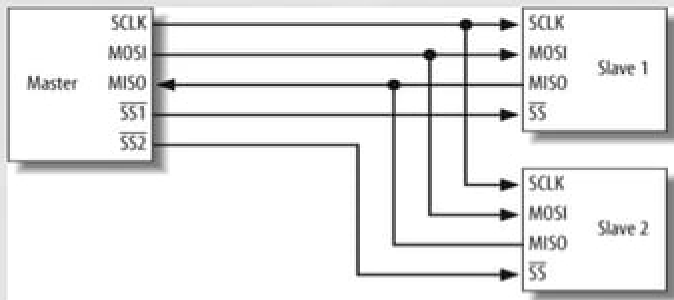
- Compares voltage between pins AIN0 and AIN1
- Asserts AC0 when AIN0 > AIN1
- AC0 can trigger timer capture function
  - Range finders indicate distance with pulse with
  - Timer capture mode can compute pulse width

Serial Peripheral Interface

- Industry standard serial protocol for communication between local devices
- Master/Slave protocol
- 3 Wire interface
- Slaves addressed via Slave Select (SS) inputs

SPI Bus – Signal Descriptions

SCLK	Serial Clock
MOSI	Master Out Slave In
MISO	Master In Slave Out
SS	Slave Select

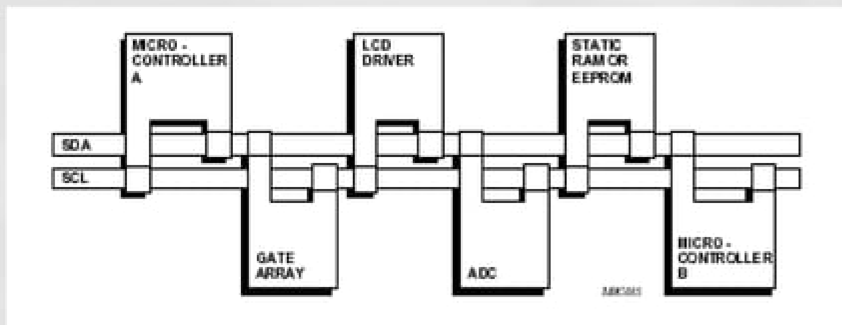


I^2C Bus Interface

- Industry standard serial protocol for communication between local devices
- Master/Slave protocol
- 2 Wire interface
- Byte oriented messages
- Slave address embedded in command

I²C Bus Signal Descriptions

SDA	Serial Data
SCL	Serial Clock



Typical SPI & I^2C Devices

- EEPROM
- IO Expanders
- Real Time Clocks
- ADC & DAC
- Temperature sensors
- Ultrasonic range finders
- Compass
- Servo / Motor Controller
- LED Display

USART

- Universal Synchronous and Asynchronous serial Receiver and Transmitter
- Full Duplex Operation
- High Resolution Baud Rate Generator
- Can provide serial terminal interface

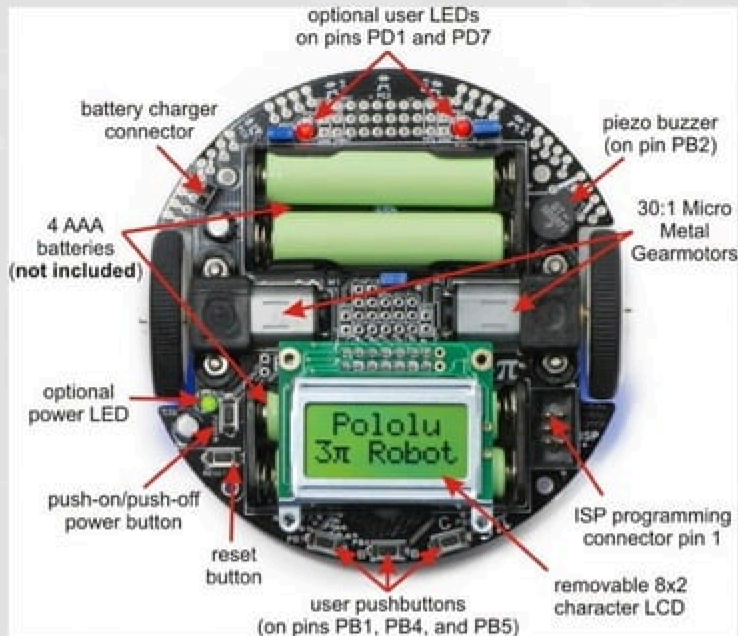
Programming Memory - JTAG

- Some chips have JTAG interface
  - Industry standard for debugging chips in circuit
  - Connect to special JTAG signals
  - Can program
    - FLASH
    - EEPROM
    - All fuses
    - Lock bits

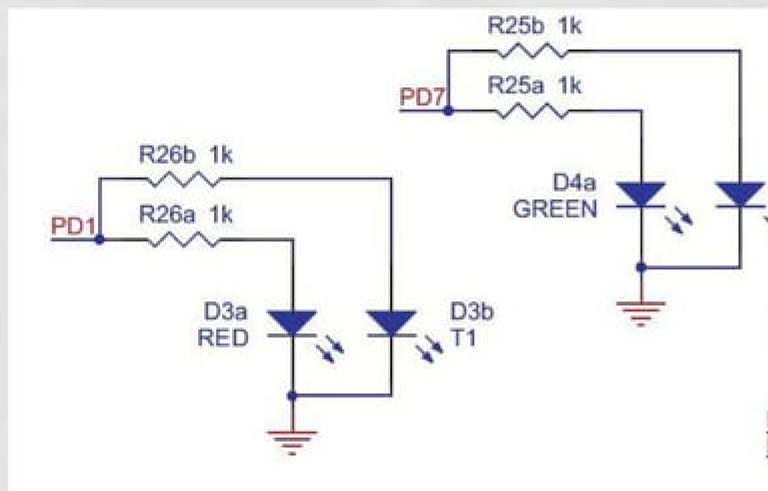
Programming Memory - ISP

- ISP (In-System Programmer)
- Connect to 6 or 10 pin connector
- SPI interface
- Special cable required
  - Can program
    - FLASH
    - EEPROM
    - Some fuses
    - Lock bits

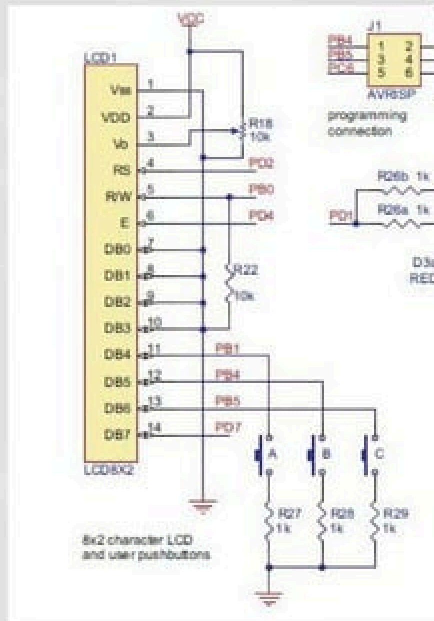
Hardware Example –Polulu 3π



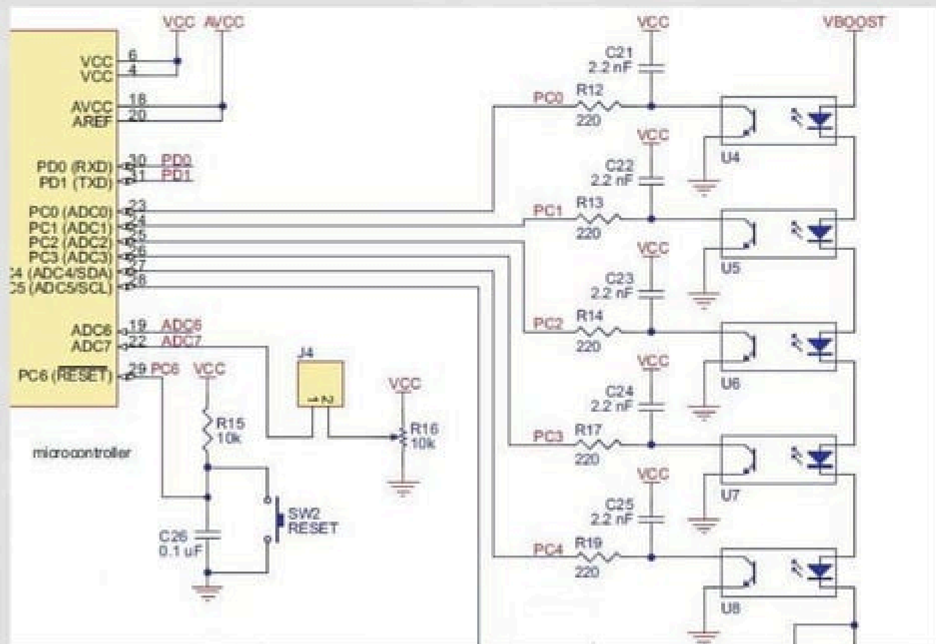
3π Port Pins: User LEDs



3π Port Pins : LCD Display & Buttons



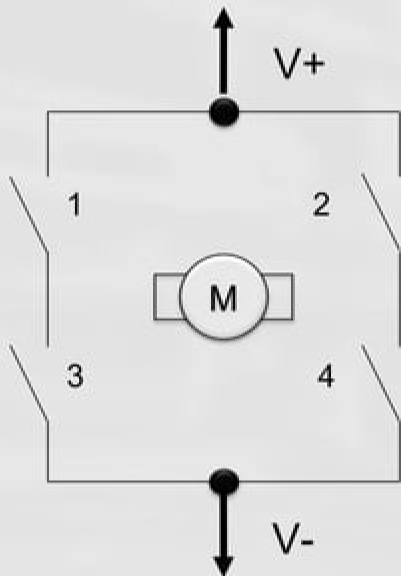
3π ADC – IR Sensors



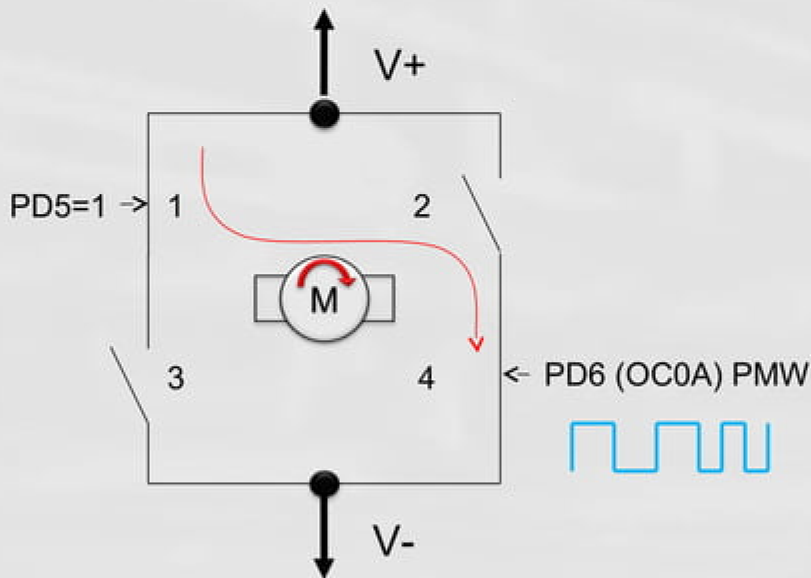
3π Motor Control

- CW: PD5 & PD6 PWM mode (OC0B, OC0A)
- CCW: PD3 & PB3 PWM mode (OC2B, OC2A)
- Motor interface uses H-Bridge controller
- Speed and direction controlled by port bits
- Controller standby controlled by PC6

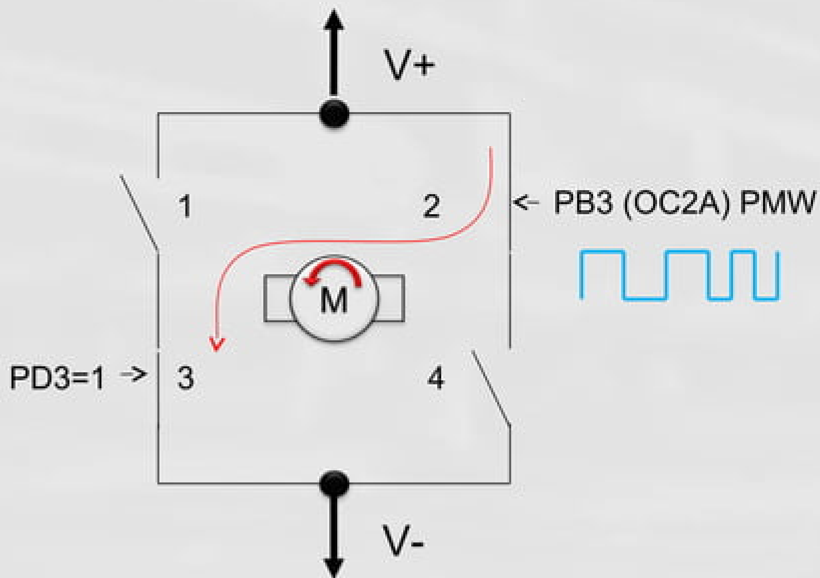
Motor Off



Motor CW



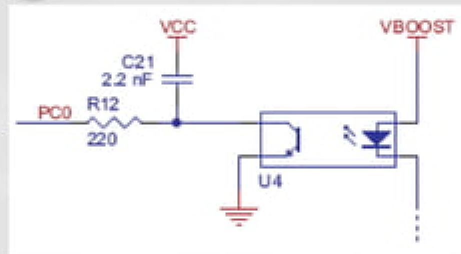
Motor CCW



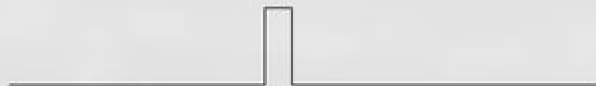
3π Reflectance Sensors

- PC5 enables the IR emitters
- One port pin for each of 5 sensors, PC0-4
- Port pin used as input and output
  - Set pin as output high for 10 μsec to charge RC circuit
  - Set pin as input
  - Measure time for voltage to drop (using TCNT2)
  - Time indicates level of light

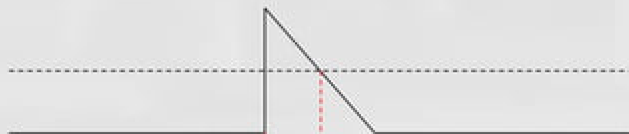
Measuring Reflectance



1. Output: 10 μsec pulse on PC0



2. Input: measure decay



t indicates amount of reflected light

3π Analog Inputs

- ADC6: battery voltage monitor
- ADC7: reads trimmer pot voltage

3π Miscellaneous

- Buzzer: Timer1 PB2 (OC1B)
- User defined (can be USART) : PD0, PD1
- Device (ISP) Programming: PB3, PB4, PB5
- 20 MHz crystal: PB6, PB7

Getting Started

- Choose a language
  - BASCOM
  - C/C++
  - Assembler (machine language)
- Choose a platform
  - Linux
  - Windows

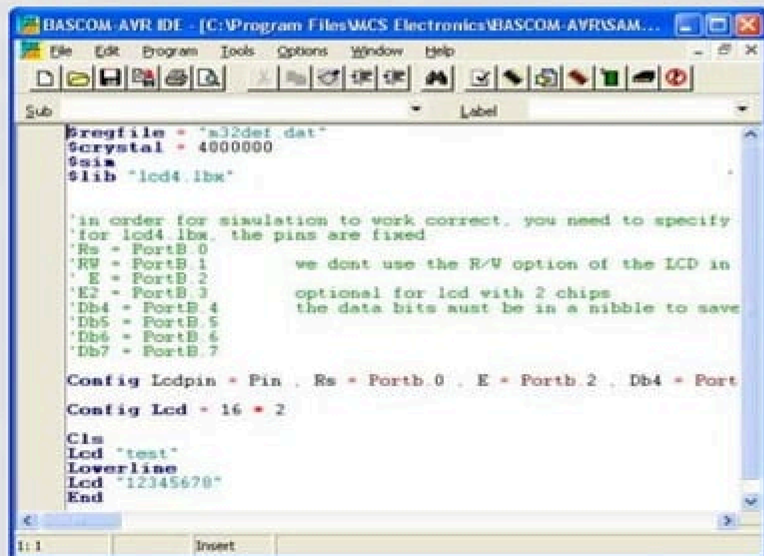
What you need

- Text Editor / IDE
- Tool Chain
  - Compiler
  - Libraries
- Programmer (cable & software)

BASCOM

- Windows based BASIC compiler for AVR Family
- Complete IDE
- Includes simulator
- Extensive library for AVR peripheral control
- Includes ISP programmer
- Demo versions available – limited code space

BASCOM IDE



BASCOM AVR Support

BASCOM-AVR Options

Compiler | Communication | Environment | Simulator | Programmer | Monitor | Printer

Chip | Output | Communication | I2C, SPI, 1WIRE | LCD

LCD type 16 * 2

BUS mode
- 4-bit
- 8-bit

Data mode
- pin
- bus

LCD-address C000

RS-address 8000

Make upper 3 bits 1 in LCD designer

Enable	PORTB.3
RS	PORTB.2
DB7	PORTB.7
DB6	PORTB.6
DB5	PORTB.5
DB4	PORTB.4

Default | Ok | Cancel

C / C++ / Assembler Tool chain

- Linux
  - AVR-GCC
    - Based on GNU toolset
    - Open Source AVR-LIBC libraries
- Windows
  - WIN-AVR
    - Port of AVR-GCC to Windows

IDEs

- Linux
  - Editors & command line tools
  - Eclipse IDE
- Windows
  - Atmel AVR Studio
    - Includes ISP programming support
  - Eclipse IDE

Other tools

- STK 200/300/400/500
  - Atmel AVR starter kit and development system
  - Interfaces to AVR Studio
- avrdude
  - Programming support for all memories

Polulu Recommended Tools

- Win-AVR
- AVR-Studio
- Proprietary libraries

Software Example

3π PID Line Follower

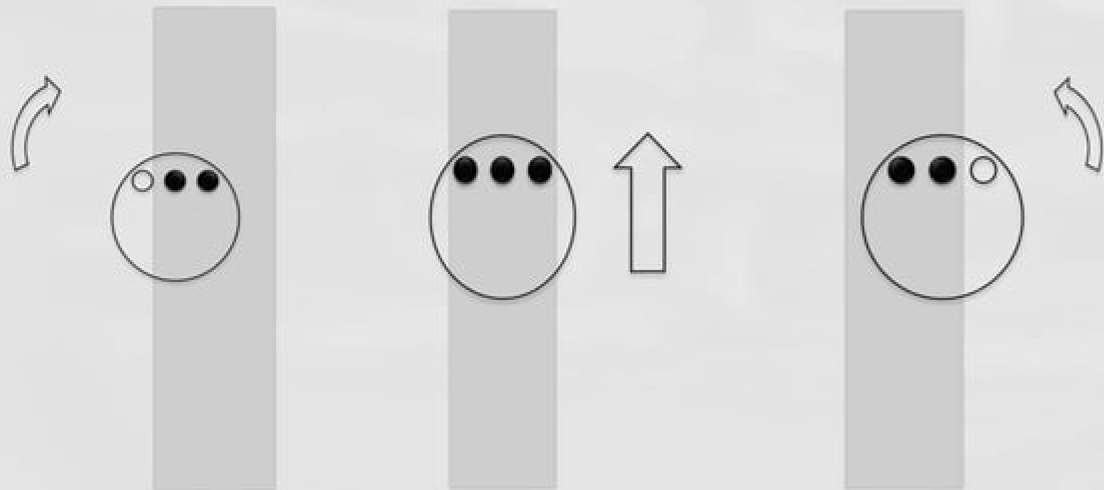
- PID Algorithm
- PID Code Walkthrough

Why PID?

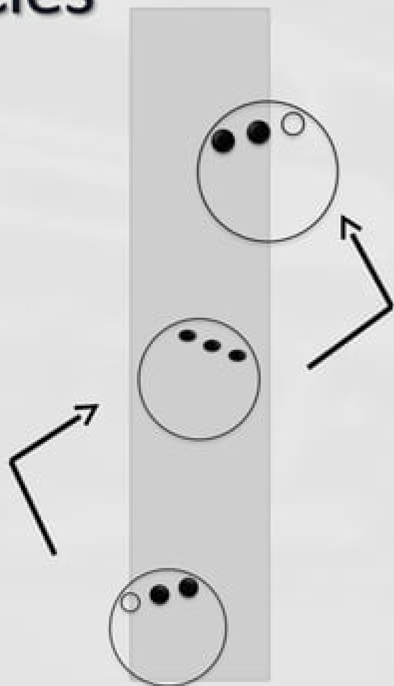
- Simple Line Follower
- Three sensors can provide position information
  - left of center
  - on center
  - right of sensors
- React to current position only

What to do?

- Three choices:



Deficiencies

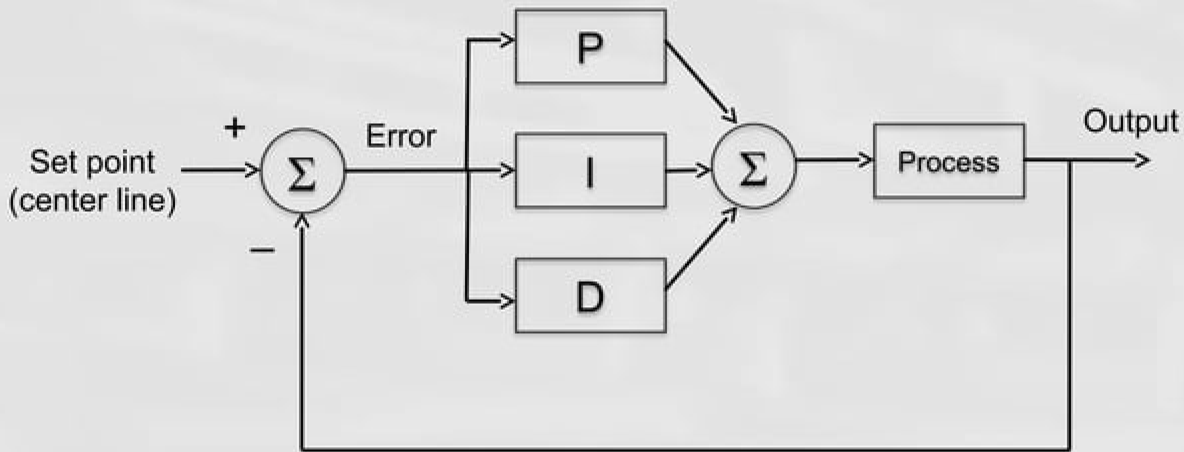


Guaranteed to zig-zag back and fort h

Better – PID

- Anticipates where we want to go
- Continuously adjusts direction to get there
- Adjustable constants control behavior

PID Algorithm



PID Components

- (P) Proportional
  - Where we are relative to where we want to be.
- (I) Integral
  - Sum of previous errors. History of where we were.
- (D) Derivative
  - Rate of change. Affects how fast we react to changes.

3π PID Implementation

- Useful functions
  - read_line
    - Returns a value between 0 – 4000
      - 0 – 1000: robot is far right of line
      - 1000 – 3000: robot is approximately centered
      - 3000 – 4000: robot is far left of line
  - set_motors
    - Set speed and direction of both motors
      - -255 – 0 : reverse direction
      - 0 – 255: forward direction

3π PID Implementation

PROGMEM Example

```
// Introductory messages.  The "PROGMEM" identifier causes the data to
// go into program space.
const char welcome_line1[] PROGMEM = " Pololu";
const char welcome_line2[] PROGMEM = "3\xf7 Robot";
const char demo_name_line1[] PROGMEM = "PID Line";
const char demo_name_line2[] PROGMEM = "Follower";

// A couple of simple tunes, stored in program space.
const char welcome[] PROGMEM = ">g32>>c32";
const char go[] PROGMEM = "L16 cdegreg4";A
```

3π PID Implementation

Sensor calibration

```
// Auto-calibration: turn right and left while calibrating the
// sensors.
for(counter=0;counter<80;counter++)
{
        if(counter < 20 || counter >= 60)
                set_motors(40,-40);
        else
                set_motors(-40,40);

        // This function records a set of sensor readings and keeps
        // track of the minimum and maximum values encountered.  The
        // IR_EMITTERS_ON argument means that the IR LEDs will be
        // turned on during the reading, which is usually what you
        // want.
        calibrate_line_sensors(IR_EMITTERS_ON);

        // Since our counter runs to 80, the total delay will be
        // 80*20 = 1600 ms.
        delay_ms(20);
}
set_motors(0,0);
```

3π PID Implementation

Compute PID values

```
// Get the position of the line.  Note that we *must* provide
// the "sensors" argument to read_line() here, even though we
// are not interested in the individual sensor readings.
unsigned int position = read_line(sensors,IR_EMITTERS_ON);
// The "proportional" term should be 0 when we are on the line.
int proportional = ((int)position) - 2000;

// Compute the derivative (change) and integral (sum) of the position.
int derivative = proportional - last_proportional;
integral += proportional;

// Remember the last position.
last_proportional = proportional;

// Compute the difference between the two motor power settings,
// m1 - m2.  If this is a positive number the robot will turn
// to the right.  If it is a negative number, the robot will
// turn to the left, and the magnitude of the number determines
// the sharpness of the turn.
int power_difference = proportional/20 + integral/12000 + derivative*7/4;
```

3π PID Implementation

Set motor speeds

```
// Compute the actual motor settings.  We never set either motor
// to a negative value.
const int max = 250;
if(power_difference > max)
          power_difference = max;
if(power_difference < -max)
          power_difference = -max;

// One Motor will always be full speed

if(power_difference < 0)
          set_motors(max+power_difference, max);
else
          set_motors(max, max-power_difference);
```

Arduino

- Open source elections prototyping platform
- Hardware
  - Based on Atmega (328 & others)
- Software
  - Wiring language, based on c++
  - Boot loader
  - Arduino IDE

ATmega Families

- ATmega 48/88/168/328
  - What we have been talking about
- ATmega 164/324/644/1284
  - JTAG interface
  - All 4 IO Ports A,B,C & D
  - More memory

ATmega Packages

- PDIP – Plastic Dual In-line Package
  - Good for hobbyists
- TQFP – Thin Quad Flat Pack
  - Surface mount
- MLF – MicroLeadFrame
  - 28 & 32 pin
  - Surface mount / higher temperature