Lecture #19

Digital To Analog, PWM, Stepper Motors

18-348 Embedded System Engineering

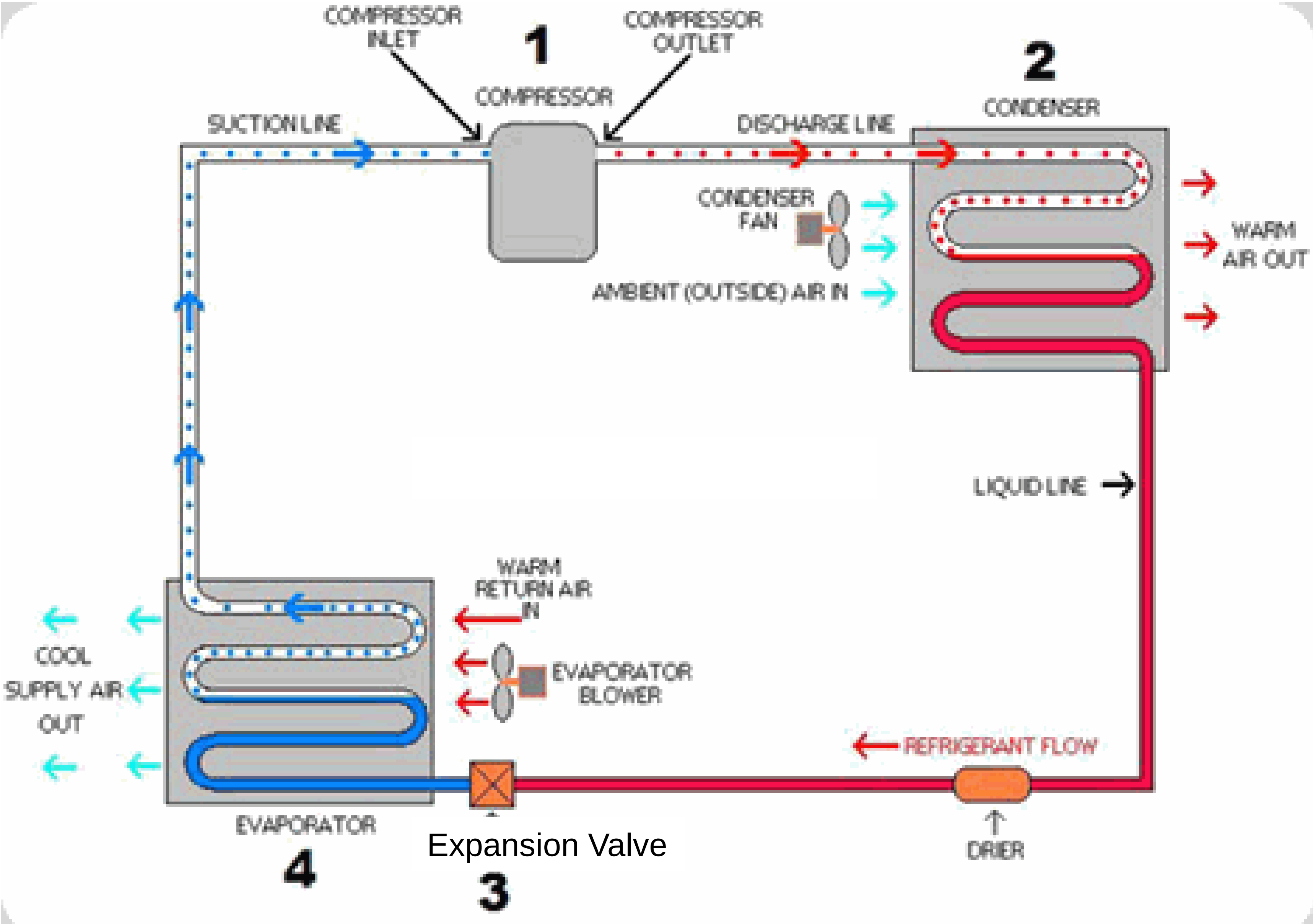
Philip Koopman

Monday, 28-March-2016

Electrical & Computer ENGINEERING

Carnegie Mellon

Example System: HVAC Compressor



[http://www.solarpowerwindenergy.org]

HVAC Embedded Control

- **Compressors (reciprocating & scroll)**
  - Smart loading and unloading of compressor
    - Want to minimize motor turn on/turn off cycles
    - May involve bypassing liquid so compressor keeps running but doesn't compress
  - Variable speed for better output temperature control
  - Diagnostics and prognostics
    - Prevent equipment damage (e.g., liquid entering compressor, compressor stall)
    - Predict equipment failures (e.g., low refrigerant, motor bearings wearing out)

- **Expansion Valve**
  - Smart control of amount of refrigerant evaporated
    - Often a stepper motor
  - Diagnostics and prognostics
    - Low refrigerant, icing on cold coils, overheating of hot coils

- **System coordination**
  - Coordinate expansion valve and compressor operation
  - Coordinate multiple compressors
  - Next lecture – talk about building-level system level diagnostics & coordination

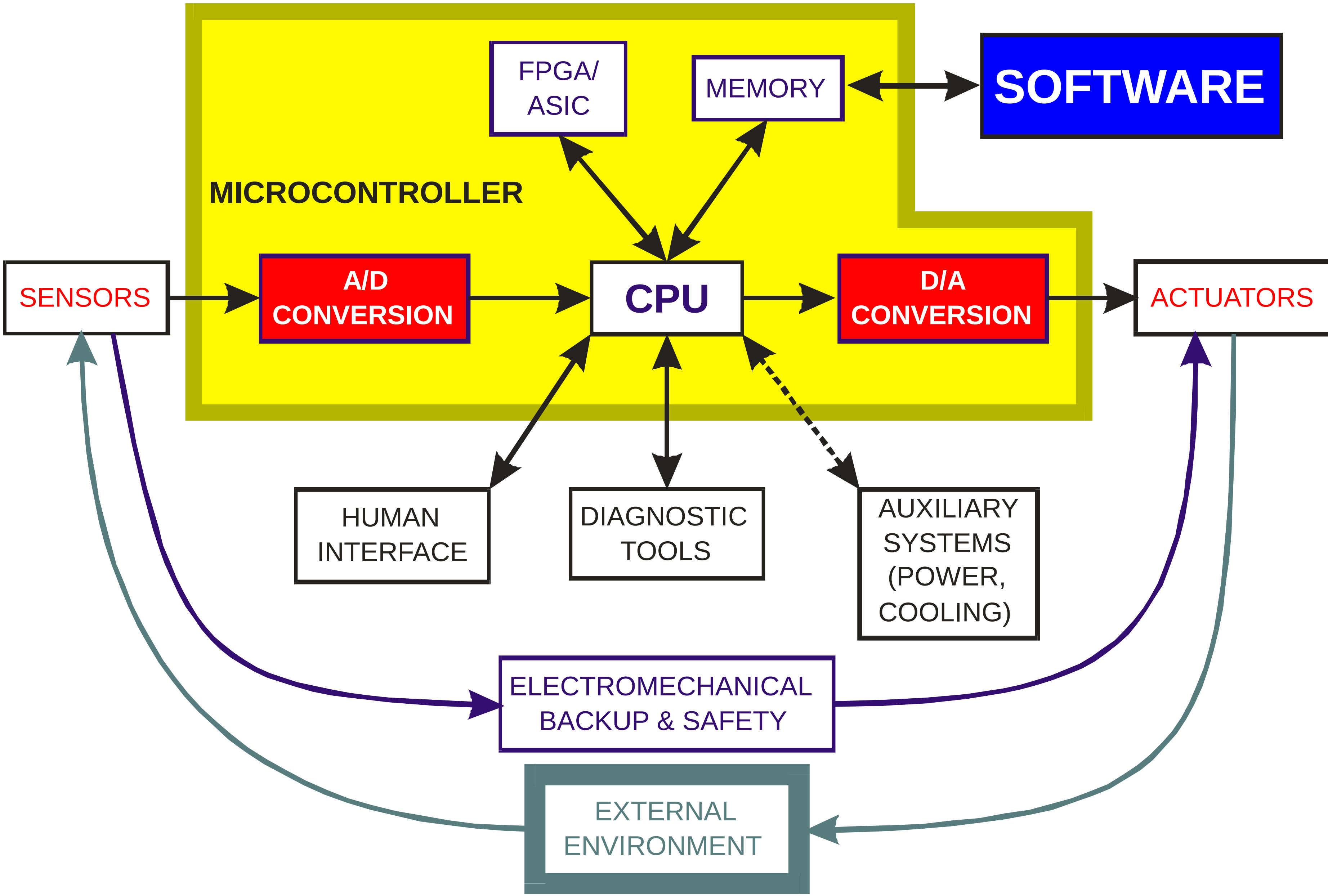
Where Are We Now?

- **Where we've been:**
  - Interrupts, concurrency, scheduling, RTOS
- **Where we're going today:**
  - Analog Output
- **Where we're going next:**
  - Analog Input
  - Human I/O
  - Very gentle introduction to control
  - …
  - Test #2 and last project demo

Preview

- **Digital To Analog Conversion**
  - Example implementation
  - Understanding performance
  - Low pass filters
- **Waveform encoding**
- **PWM**
  - Digital way to “fake” analog
  - How to use course processor PWM support hardware
  - How a servo works
- **How a stepper motor works**
  - Note: 3-D printers are mostly stepper motors + PWM

Big Picture – I/O Is Where The Work Gets Done!



Analog ➔ Digital ➔ Analog Conversion

- **The real world is analog**
  - Voltage, current are continuous
  - Time is continuous
- **But the computing world is discrete**
  - Bits, bytes
  - Some sensors/actuators use digital values…
    … but many deal with analog values

- **A/D conversion** “analog to digital”
  - Getting analog inputs to digital form

- **D/A conversion** “digital to analog”
  - Getting digital inputs to analog form

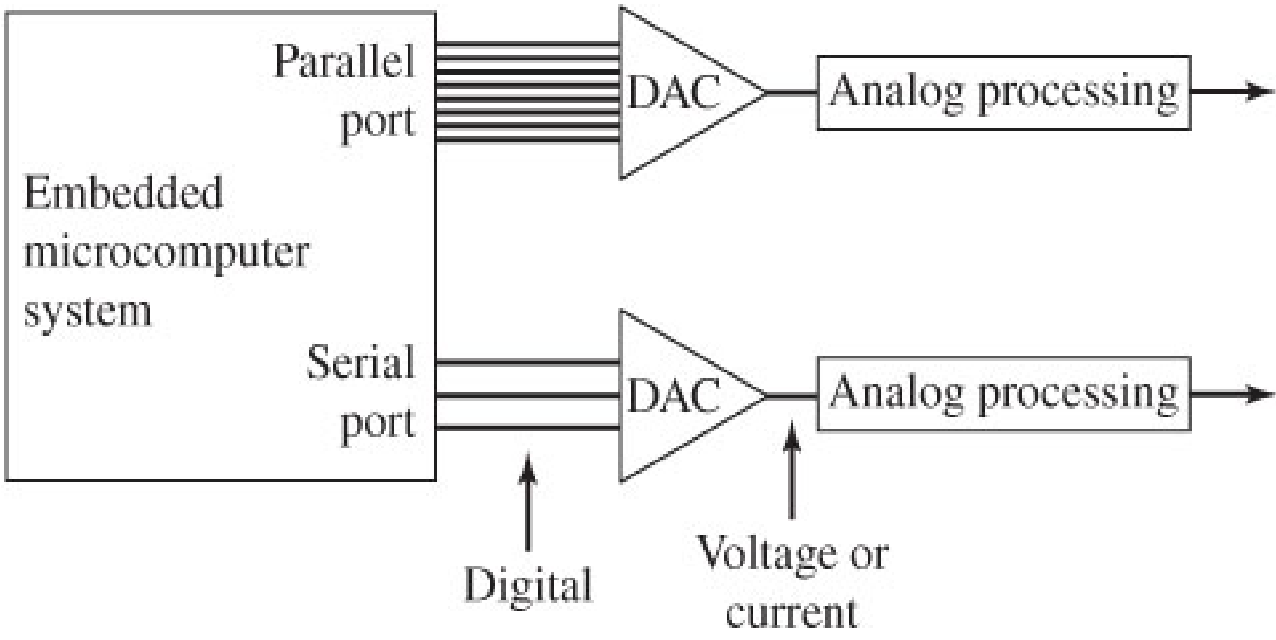
- **Digital I/O**
  - Sometimes you can fake analog values with digital (e.g., digital pulsing)

D/A Conversion

- **"DAC"** = "D/A Converter" = "Digital To Analog Converter"
- **Given several bits of a digital value, convert it to an analog value**
  - Usually voltage or current
  - Many drives an actuator, further converting output into motion, heat, light, etc.
  - Might be directly connected to CPU or accessed via a serial bus

Figure 11.33
DACs provide analog output for our embedded microcomputer systems.

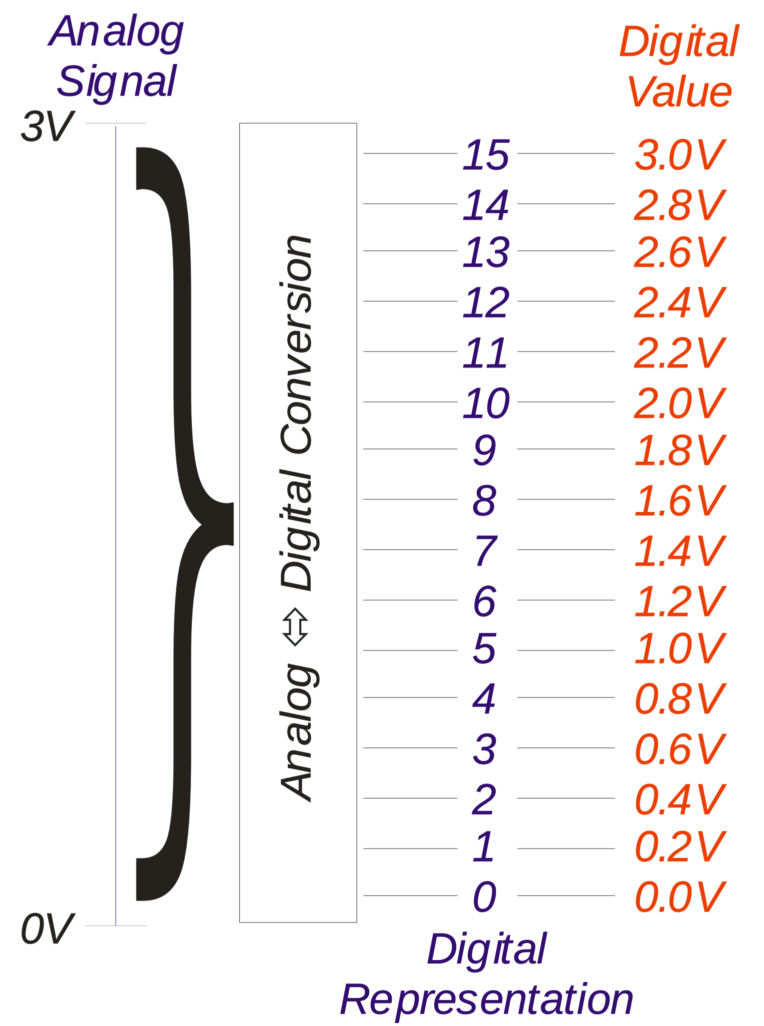
[Valvano]



General Idea Of A DAC

- **Input is bits**
  - k-bit value,   often 8 bits but can be any integer number
  - Signed or unsigned number (often unsigned)

- **Output is an analog value (volts, amps)**
  - Digital value determines output
  - Can be output many ways:
    - – Absolute voltage
    - – Offset added to reference voltage
    - – Current (mAmps, Amps)



Analog Circuits Aren't "Ideal"

- Real DACs have offset error, gain error, slew, ringing, nonlinearities…

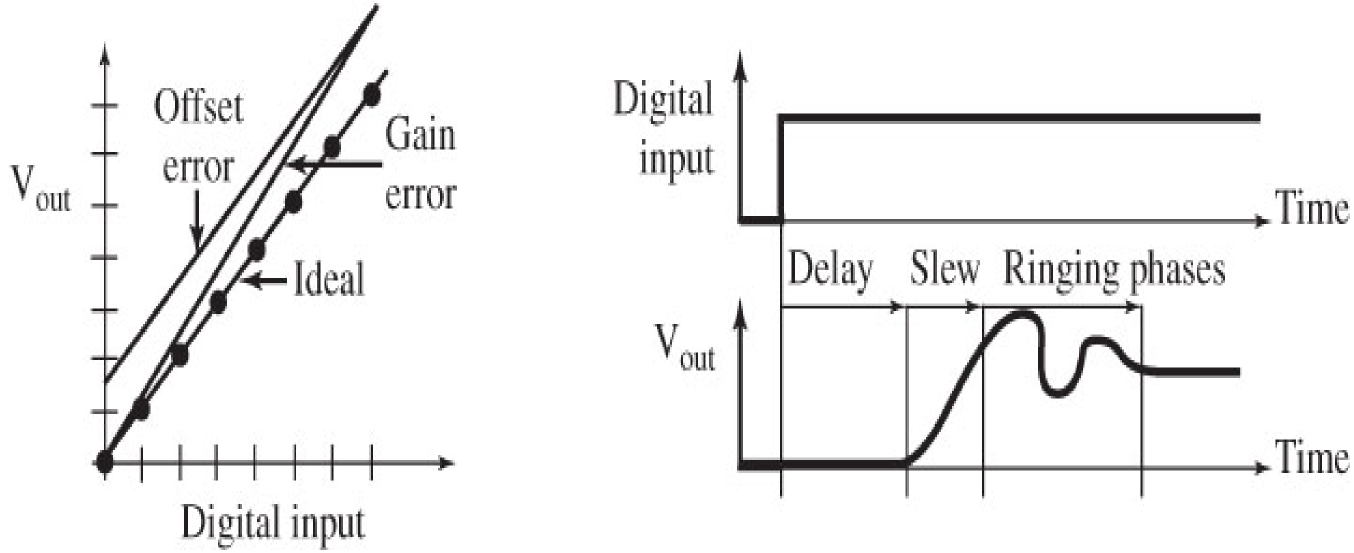


Figure 11.35
Static and dynamic performance measures of DACs.

[Valvano]

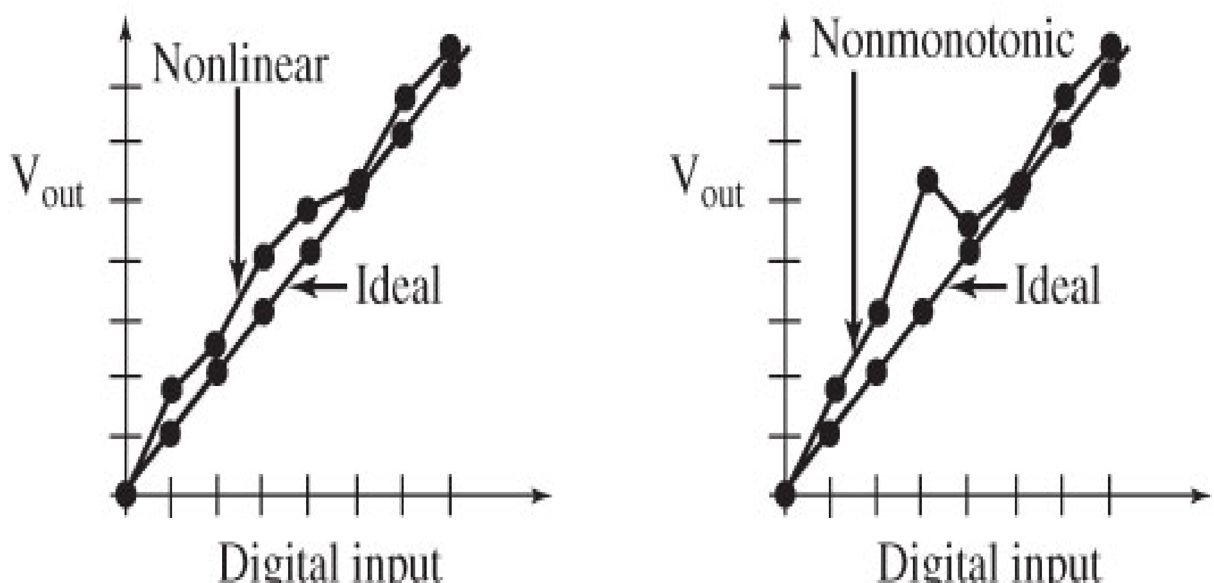


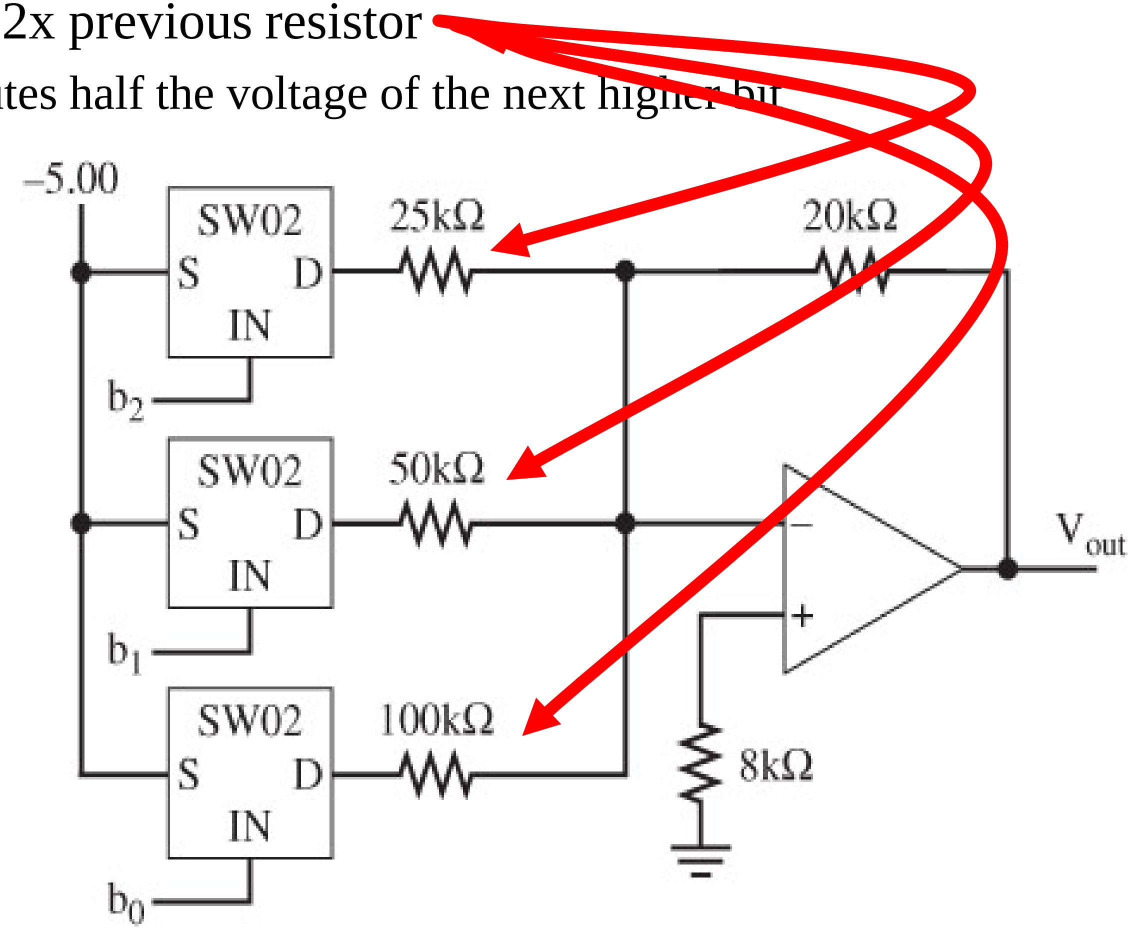
Figure 11.36
Nonlinear and nonmonotonic DACs.

[Valvano]

Example 3-Bit DAC

- **Operating Principle for "Summing DAC":**
  - One switch per bit; bit closes switch when "on"; opens switch when "off"
  - Different resistor values put different voltages out
  - Net resistance (hence input voltage) subject to parallel resistance value
  - Note: each resistor 2x previous resistor
    - Each bit contributes half the voltage of the next higher bit

Figure 11.38 Three-bit unsigned summing DAC.



[Valvano]

DAC Performance

- **Usually, DACs attempt to be linear:**

$$V_{out} \approx V_{fullscale}\left(\frac{b_7}{2}+\frac{b_6}{4}+\frac{b_5}{8}+\frac{b_4}{16}+\frac{b_3}{32}+\frac{b_2}{64}+\frac{b_1}{128}+\frac{b_0}{256}\right)+V_{offset}$$

  - Notes: $V_{fullscale}$ input in this equation has to be 1/256 above output "full scale"
    - If all these bits are on, result is 255/256 of $V_{fullscale}$
  - V_{offset} is supposed to be zero in most applications
  - Doesn't take into account non-ideal behaviors!

- **Quantization effects – value**
  - Analog value isn't exact
  - Analog value is approximated via a "bin" or voltage quantum
  - Bin size is ~$1/2^K$ of full scale (not quite because of the "fencepost" numbering issue)

- **Quantization effects – time**
  - Analog value produced periodically by CPU
  - Not continuously as with real analog signal!

Generating An Analog Waveform – Computed

- **Periodic output values**
  - Use timer-based interrupt
  - (What is the problem with this particular example from Valvano?)

Program 11.2
Periodic interrupt used to create the analog output waveform.

```
unsigned short wave(unsigned short t){
        float result,time;
        time=2*pi*((float)t)/1000.0;
// integer t in msec into floating point time in seconds
        result=2048.0+1000.0*cos(31.25*time)-500.0*sin(125.0*time);
        return (unsigned short) result;}
```

```
#define Rate 2000
#define OC5  0x20
unsigned short Time;  // Inc every 1ms
#pragma interrupt_handler TOC5handler()
void TOC5handler(void){
   TFLG1=OC5;        // ack C5F
   TC5=TC5+Rate;     // Executed every 1 ms
   Time++;
   DACout(wave(Time));}
```

[Valvano]

Generating An Analog Waveform – Table Based

// 6811
#define Rate 2000
#define OC5 0x08
#pragma interrupt_handler TOC5handler()
void TOC5handler(void){
   TFLG1=OC5; // Ack interrupt
   TOC5=TOC5+Rate; // Executed every 1 ms
   if((++I)==32) I=0;
   DACout(wave[I]);}
	// 6812
	#define Rate 2000
	#define OC5 0x20
	#pragma interrupt_handler TOC5handler()
	void TOC5handler(void){
	   TFLG1=OC5; // ack C5F
	   TC5=TC5+Rate; // Executed every 1 ms
	   if((++I)==32) I=0;
	   DACout(wave[I]);}

Program 11.5
Periodic interrupt used to create the analog output waveform.

Program 11.4
Simple data structure for the waveform.

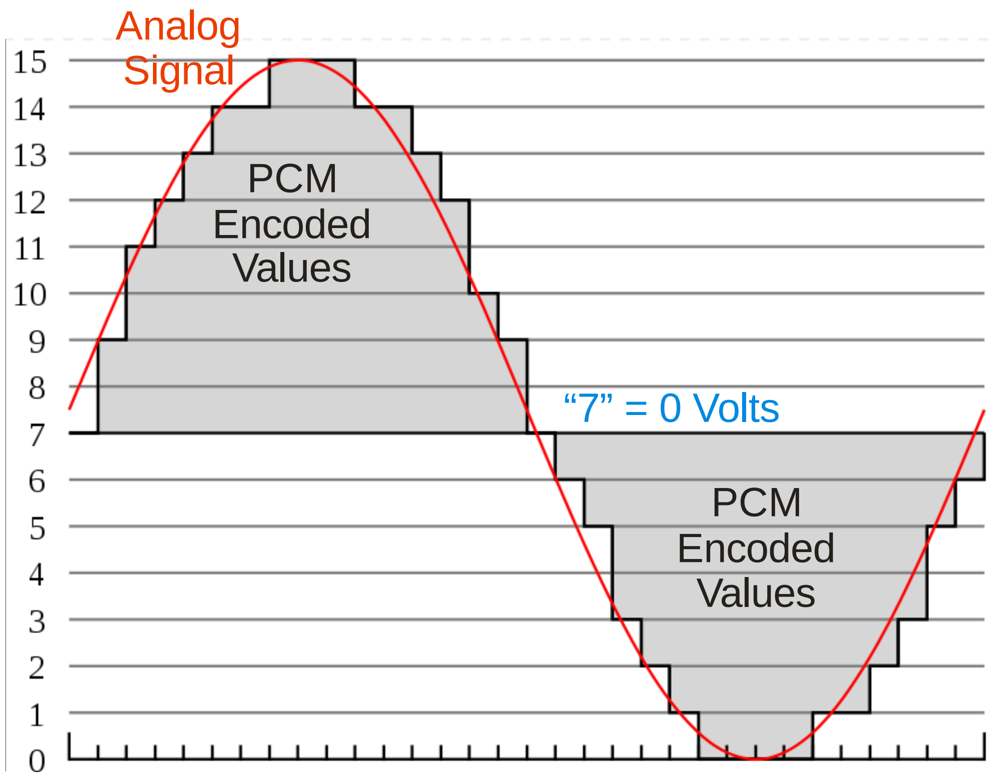
[Valvano]

```
unsigned short I;  // incremented every 1ms
const unsigned short wave[32]= {
       3048,2675,2472,2526,2755,2957,2931,2597,
       2048,1499,1165,1139,1341,1570,1624,1421,
       1048,714,624,863,1341,1846,2165,2206,2048,
       1890,1931,2250,2755,3233,3472,3382};
```

Encoding Waveforms – PCM

- **Sample rate – how often are the samples?**
  - Want samples at least twice as fast as highest frequency component (a.k.a. Nyquist Rate)

- **PCM – Pulse Code Modulation**
  - Use the binary value in each sample
  - Use as an unsigned value
    - Typically put zero point in middle
    - E.g. 0-15; 7 = 0 Volts
0 = -5 Volts
  - Example:
CD-Audio is 16-bit PCM at 44 KHz (stereo – two channels)
    - Why 44 KHz?

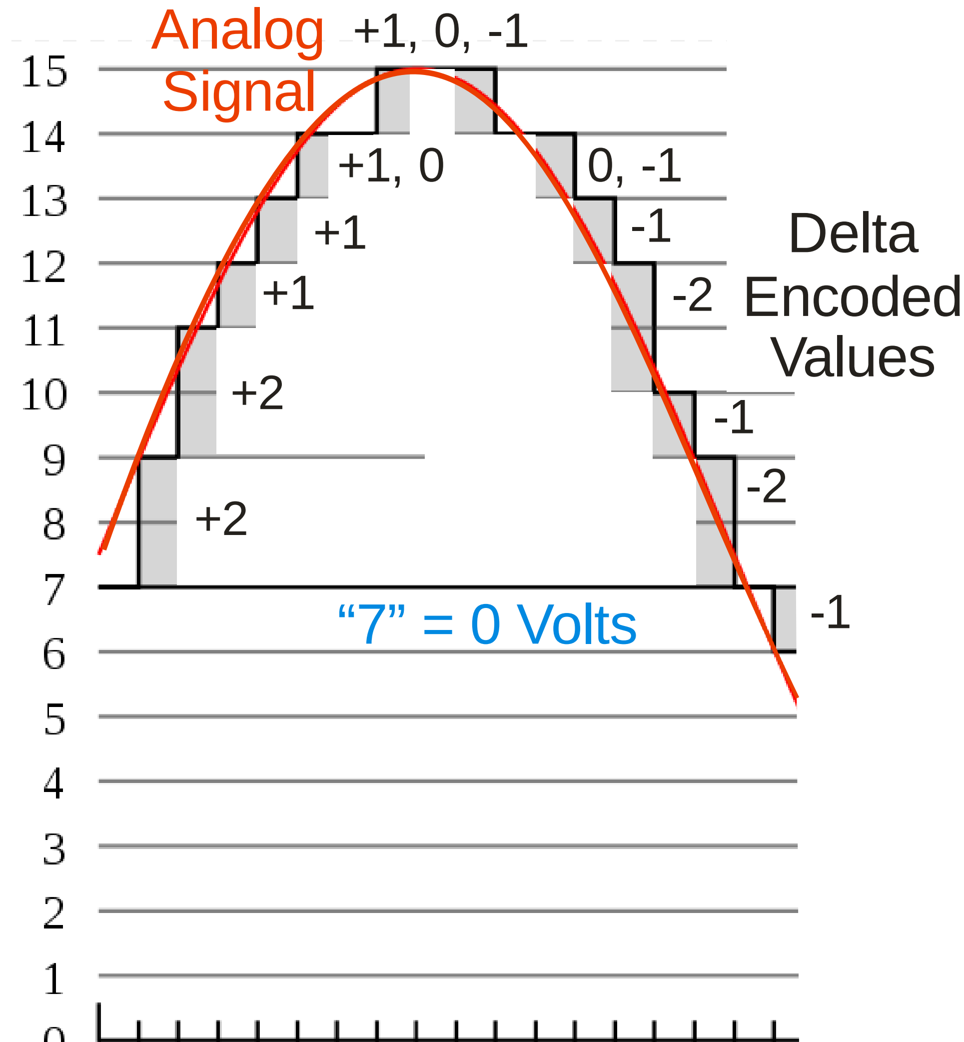


[Wikipedia:PCM], *modified*

Delta Modulation

- **Delta Modulation**
  - Use the difference from last sample
  - Uses fewer bits per sample…
    … but assumes signal changes gradually
  - Bits per sample related to bandwidth of signal – higher bandwidth means bigger deltas (more bits per sample)
  - Example on right is, perhaps, 3 bit encoding: { -4, -3, -2, -1, 0, 1, 2, 3}
  - Values are twos complement rather than unsigned
  - Values must be added to running total (i.e., integrated)

- **Other more sophisticated encodings**
  - Linear predictive coding
  - Application-specific coding (MP3, etc.)

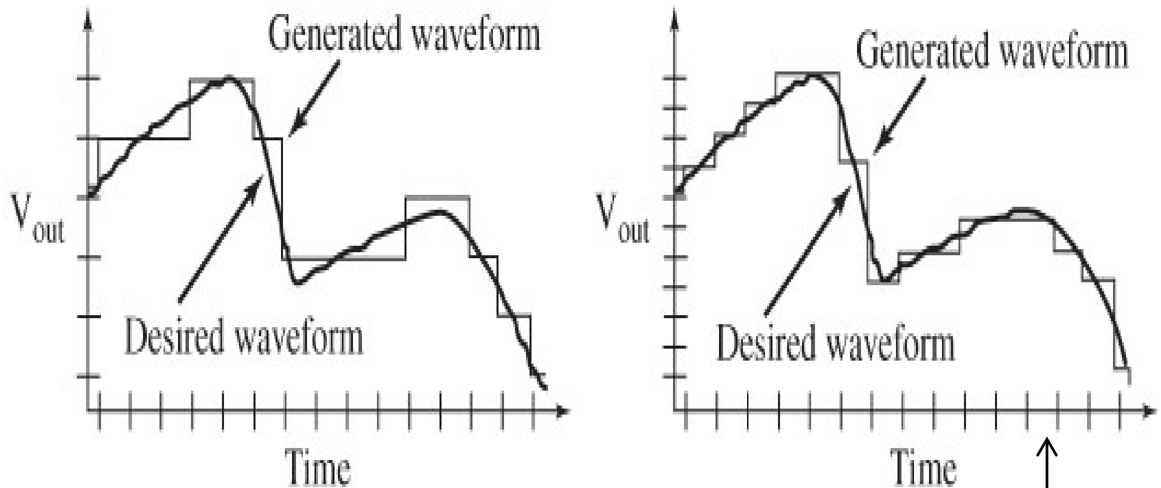


It’s All About The Bandwidth – Bits Per Second

- **Increasing # bits of resolution improves output waveform**
  - Reduces value quantization error

Figure 11.46
The waveform on the right was created by a DAC with one more bit than the left.

[Valvano]

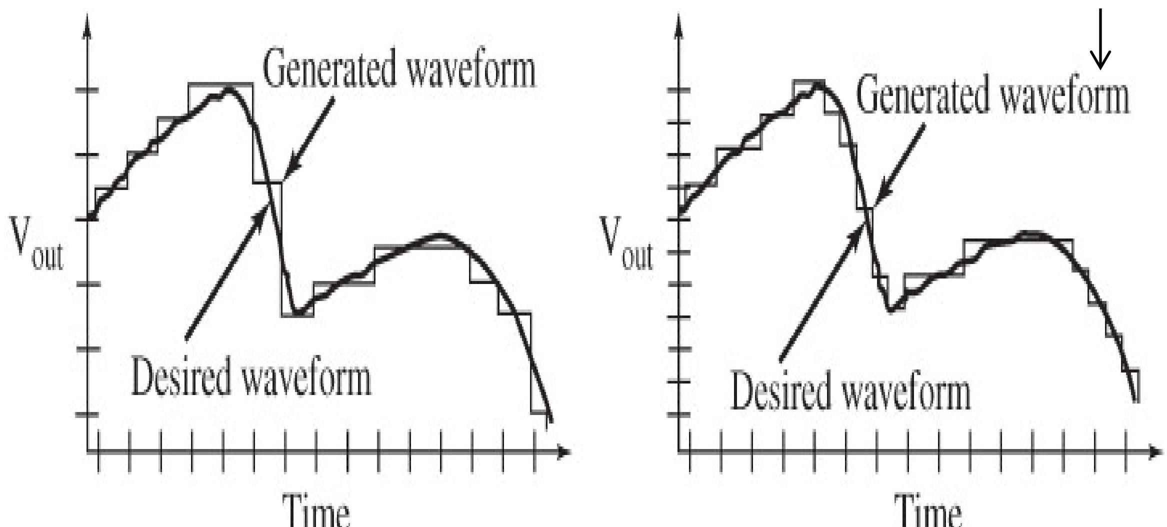


- **Increasing sample rate improves output waveform**
  - Reduces time quantization error

PICTURES REVERSED

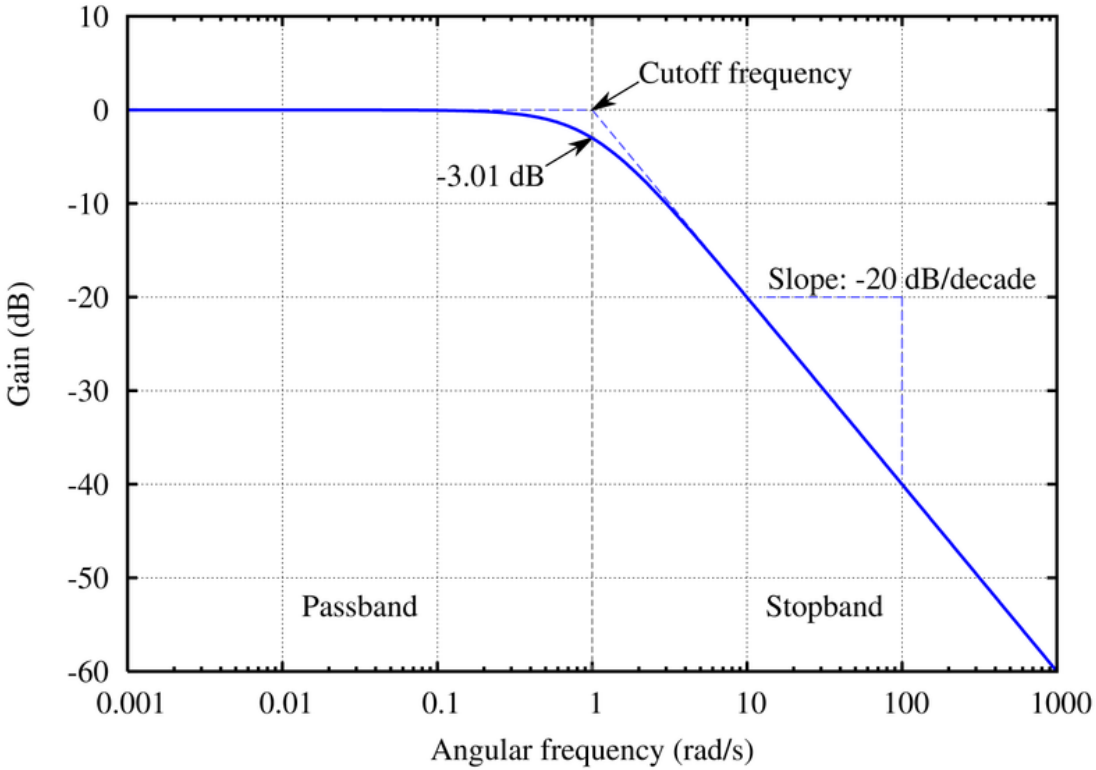
Figure 11.47
The waveform on the right was created by a system with twice the output rate than the left.

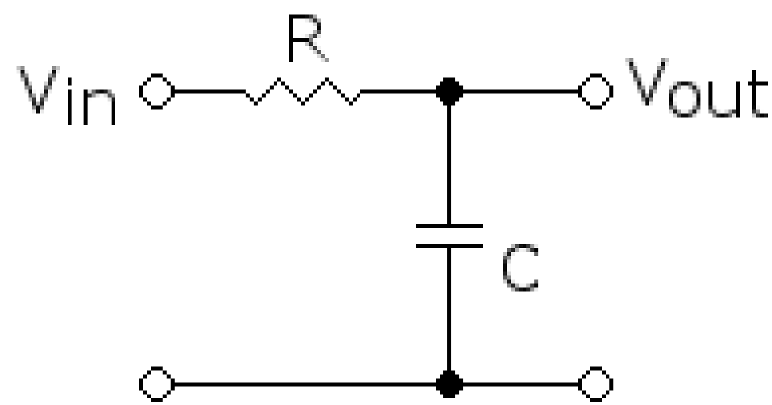
[Valvano]



Low Pass Filters

- **Can we get rid of the bumps in the output?**
  - Add more bits (expensive, doesn't necessarily work very well)
    ..OR..
  - Use a Low Pass Filter!





[Wikipedia]

- **Or, sometimes … do nothing (implicit low pass filter)**
  - Physical time constants of controlled system or actuators might smooth bumps!

Pulse Modulation

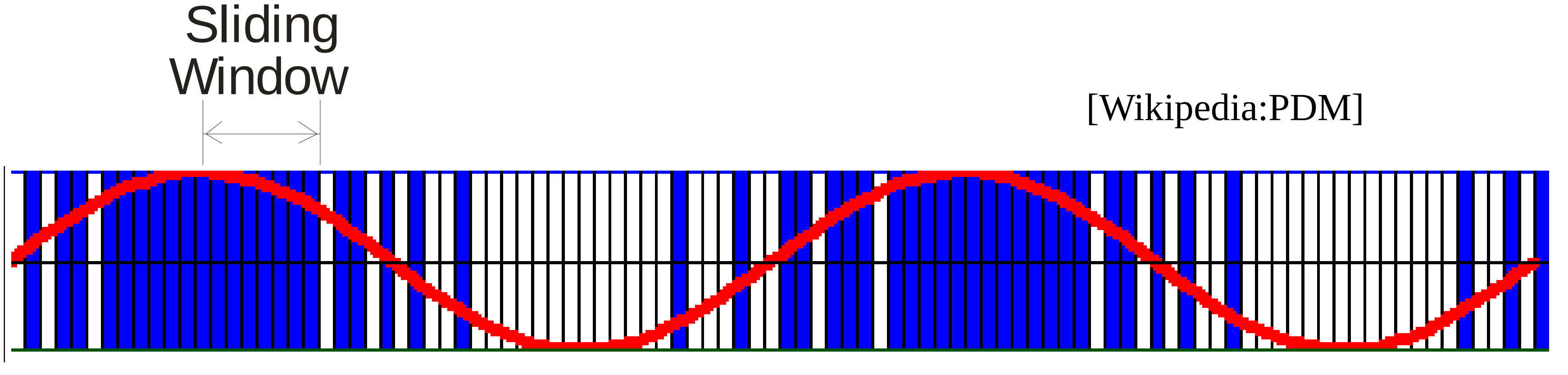
- **DACs are expensive – take a lot of area**
  - And even more if you want lots of analog output channels!
    - The course processor doesn't even have D/A outputs built into it
  - So, how do you actually do D/A conversion without a DAC?
  - Preferably using a *single output pin*?
  - Preferably in a way that is lower noise than a DAC (e.g., purely digital)

- **Can use purely digital output to "fake" analog output**
  - Pulse Density Modulation
    - Use high speed bit stream to represent proportion of full scale value
  - Pulse Width Modulation
    - Send varying width of pulses to change power/duty cycle of actuator
  - Others:
    - Pulse Rate modulation (how often a pulse is sent)

Pulse Density Modulation

◆ Look at a sliding window of *p* pulses

- Bit value of 1 = “+1” Bit value of 0 = “-1”
- Signal value is the sum of the +1 and -1 values of the bits in the window
- Generally want very high bit rate for this to work (used in audio systems)
  - Works on AC signals; can have offset error on slow or DC signals
- Get analog output with LP Filter (capacitor does analog work)



[Wikipedia:PDM]

◆ How do you know signal is going down just after the peak?

- When first -1 enters sliding window, output starts going down
- Output is phase-shifted to the right by the sliding window size

PDM Implementation Sketch

```
for(;;)
{ { if (<next sample time>) { <update desired_output> }
  if (desired_output > current_output)
  {  output(1);    // Go up if we are currently too low
     current_output += delta_value;
  } else
  {  output(0);    // Go down if we are currently too high
     current_output -= delta_value;
  }
  <wait for next output bit time; constant bit rate>
}
```

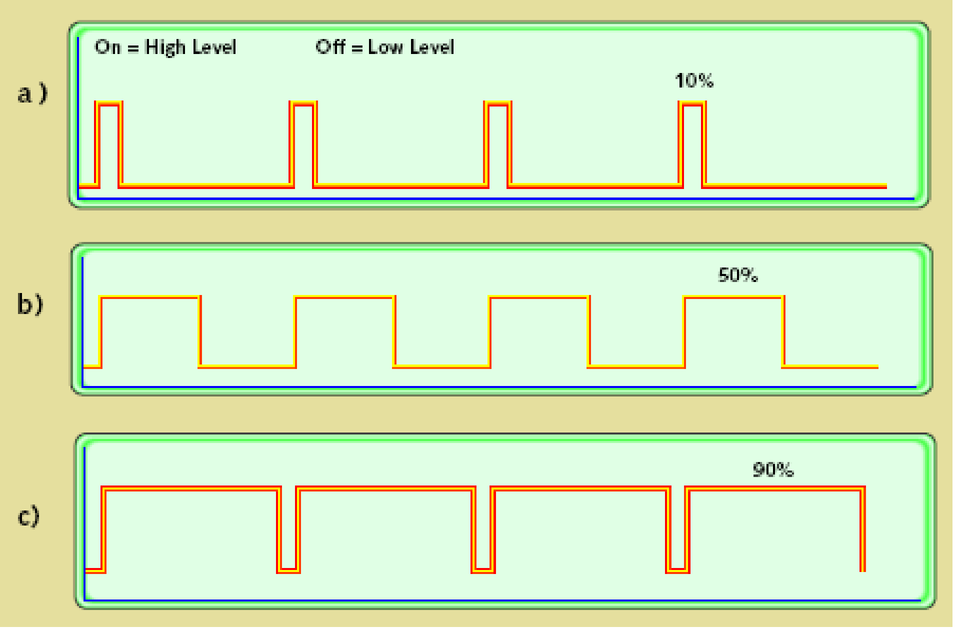
- **Tradeoffs:**
  - With only two values, analog noise less of an issue (only “hi” and “lo”)
  - Direct tradeoff of value quantization vs. time quantization
    - Big window gives more values, but takes longer to make big changes
    - Small window has less phase shift, but supports fewer total values
    - It’s all about the bandwidth – bits per second is the limiting factor

Pulse Width Modulation

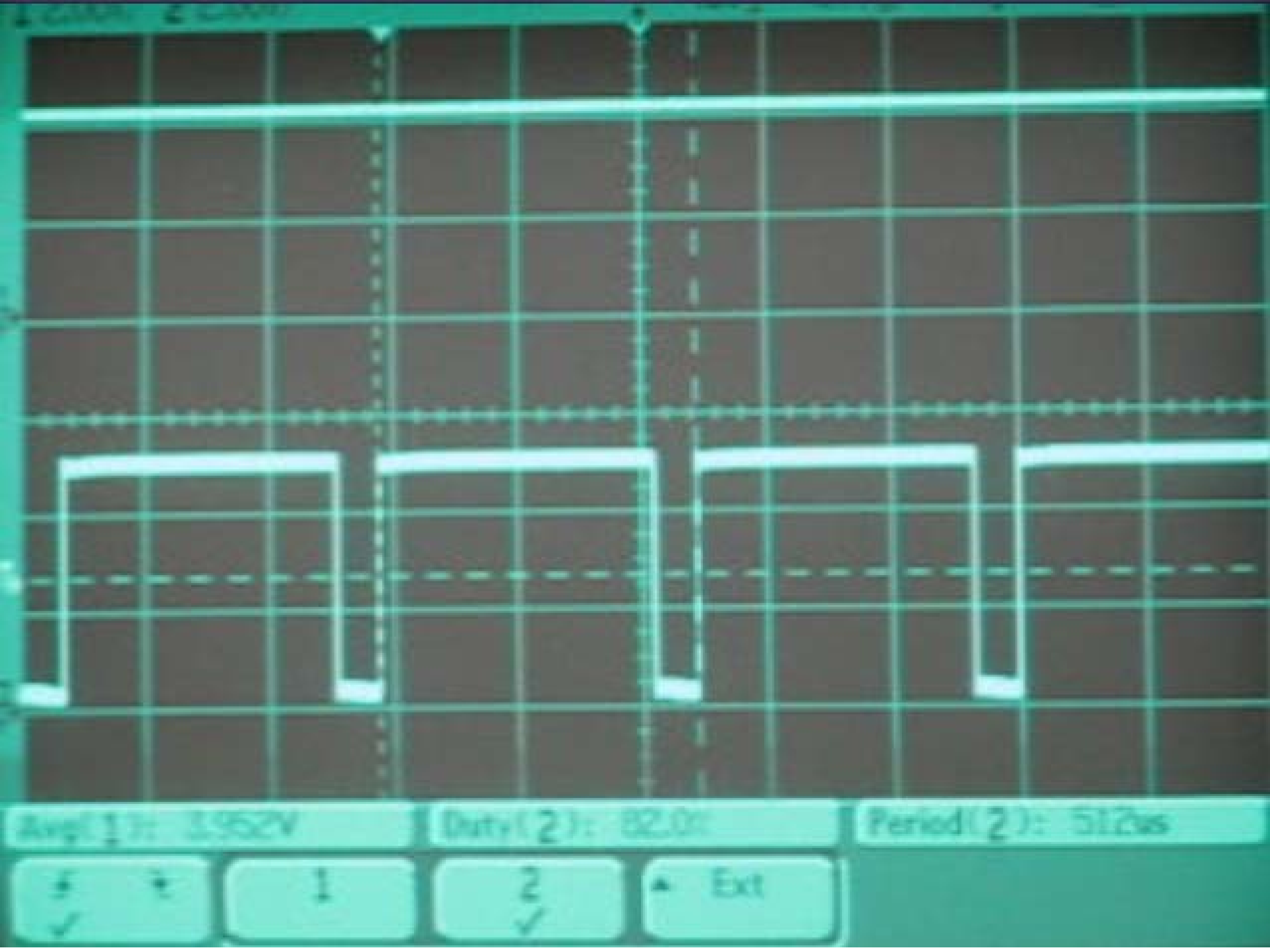
- **Idea: represent value with high/low duty cycle**
  - Constant period, some high and some low within period
  - e.g., 30% duty cycle: 3 msec high, 7 msec low (10 msec period)
  - e.g., 90% duty cycle: 90 msec high, 10 msec low (100 msec period)

- **Can be used to deliver varying levels of power**
  - This is how LED dimming works
  - Often relies on time constant of actuator to do LP Filtering "for free"
  - Can be used, for example, to control cooling fan speed
    - Physical inertia of the fan integrates pulses into an average fan speed

FIGURE 1 PWM signals of varying duty cycles



[Barr01]


Avg(1): 3.952V
Duty(2): 82.0
Period(2): 512us
Ext

PWM Block Diagram

- See Chapter 12 of MC8S12 data sheet for details

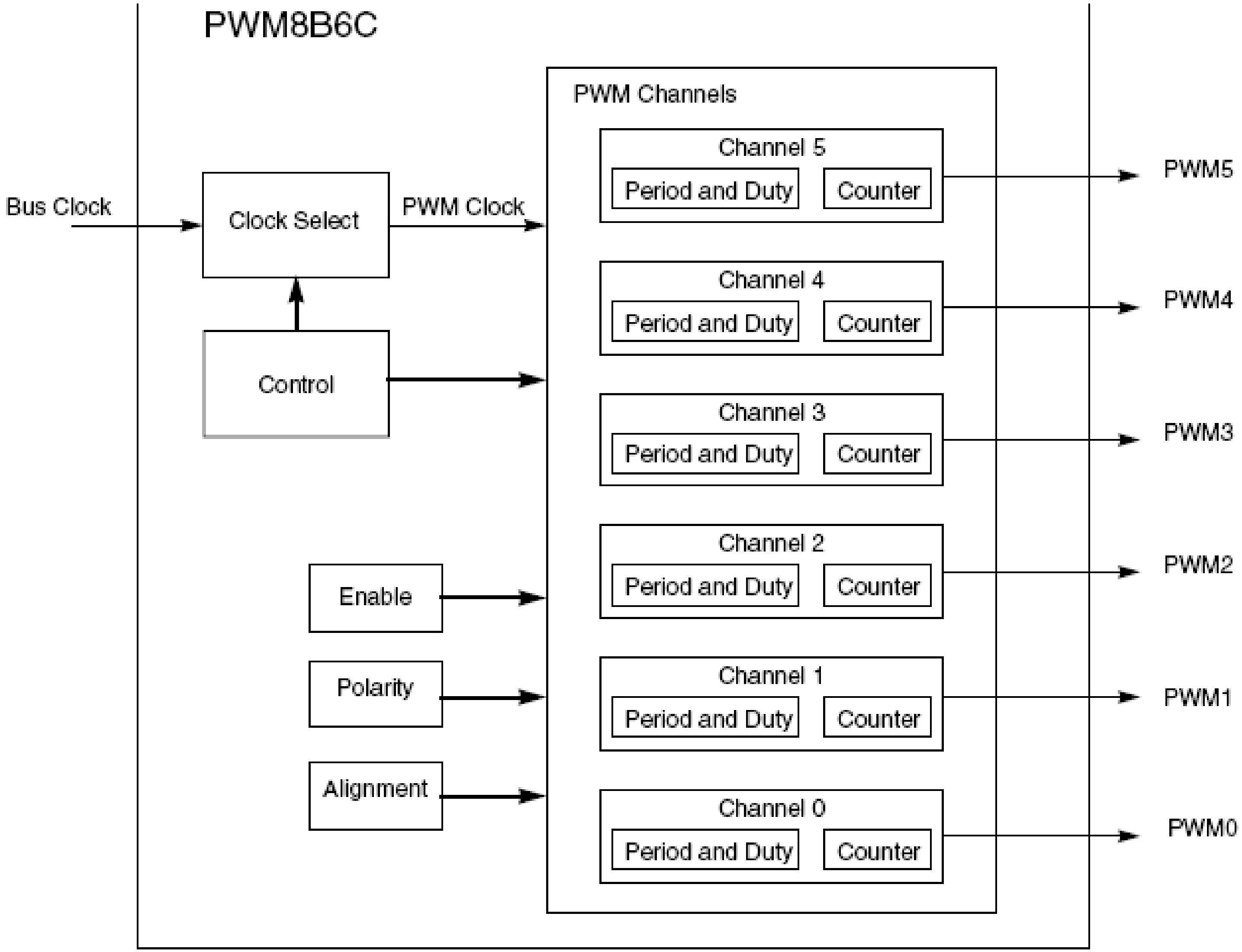
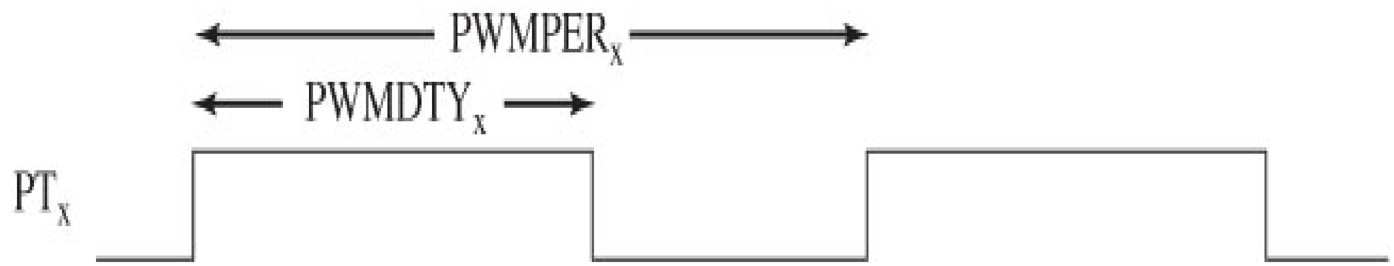


Figure 12-1. PWM8B6CV1 Block Diagram

PWM Registers

- **MODRRx – Timer vs. PWM channel x (1 = PWM)**
- **PWMEx – enable PWM channel x (1 PWM)**
- **PWMPOLx – polarity**
  - 0 = low followed by high (first part of pulse is low)
  - 1 = high followed by low (first part of pulse is high)
- **PWMPRCLK – clock prescaler (similar to other clock prescalers)**
- **PWMCLK – clock select for PWM (Clock A/B or Clock SA/SB)**
  - Clocks SA/SB are scaled versions of Clock A, Clock B
  - E.g., PWMSCLA is scaling value for Clock A – lets it run up to 512x slower
- **PWMCTL – control register concatenation**
  - Concatenates pairs of 8-bit counters to give 16-bit counters
  - CON23: channel 2 register is high-order byte of a 16-bit channel
- **PWMPERx – period for channel**
- **PWMDTYx – duty cycle for channel**
- **PWMSDN – optional pin for emergency shutdown of pulses**
  - Interrupt vector $FF8C
  - Why do you want emergency shutdown of pulses?

Figure 6.24
PWM output generated when PPOL=1.




```
;MC9S12C32 assembly
PWM_Init3            ;1s PWM on PT3
    bset MODRR,#$08  ;PT0 with PWM
    bset PWME,#$08   ;enable chan 3
    bset PWMPOL,#$08 ;high then low
    bclr PWMCLK,#$08 ;Clock B
    bset PWMCTL,#$20 ;concat 2+3
    ldaa PWMPRCLK
    anda #$8F
    oraa #$60
    staa PWMPRCLK        ;B=E/64
    movw #62500,PWMPER23 ;1s period
    movw #0,PWMDTY23     ;off
    rts
PWM_Duty3          ;RegD is duty cycle
    std  PWMDTY0   ;0 to 62500
    rts
```

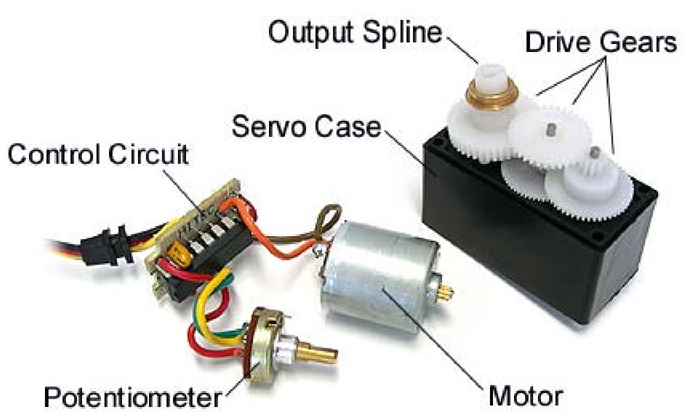
```
// MC9S12C32 C
// 1s PWM on PT3
void PWM_Init(void){
  MODRR  |= 0x08; // PT3 with PWM
  PWME   |= 0x08; // enable channel 3
  PWMPOL |= 0x08; // PT3 high then low
  PWMCLK &=~0x08; // Clock B
  PWMCTL |= 0x20; // Concatenate 2+3
  PWMPRCLK = (PWMPRCLK&0x8F)|0x60; // B=E/64
  PWMPER23 = 62500; // 1s period
  PWMDTY23 = 0;     // initially off
}
// Set the duty cycle on PT3 output
void PWM_Duty(unsigned short duty){
  PWMDTY23 = duty;  // 0 to 62500
}
```

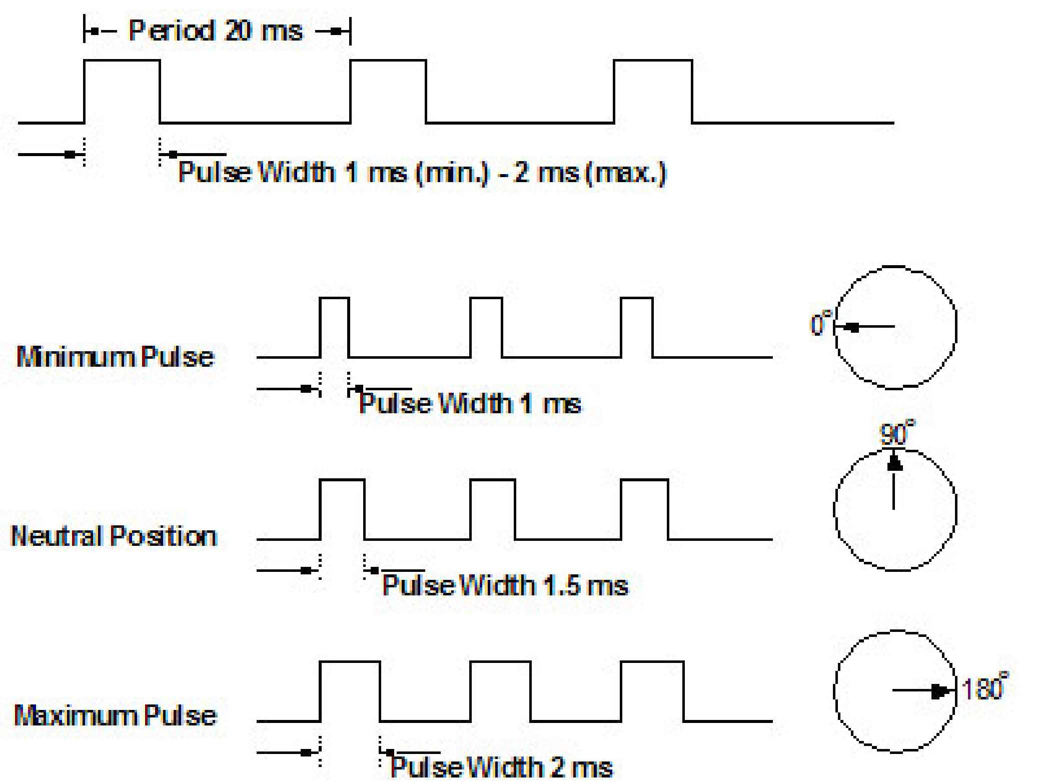
Program 6.21
Implementation of an 8-bit PWM output.

[Valvano]

How Do Servos Work?

- **Uses PWM to set position between "zero" and "full"**
  - PWM sets commanded position
  - Potentiometer used to measure actual position
  - Servo self-adjusts to keep actual position at/near commanded position
    - Closed loop control; maintains position even if external forces try to move servo





http://www.servocity.com/html/how_do_servos_work_.html

Is PWM A Free Ride?

- **Digital values have very low amplitude noise**
  - Analog values – noise shows up in any disturbance
  - Digital values – noise only if signal crosses threshold

- **Is it a free lunch?**
  - No– still have noise in *timing*
    - Clock edges can move around depending on value noise, ringing, etc.

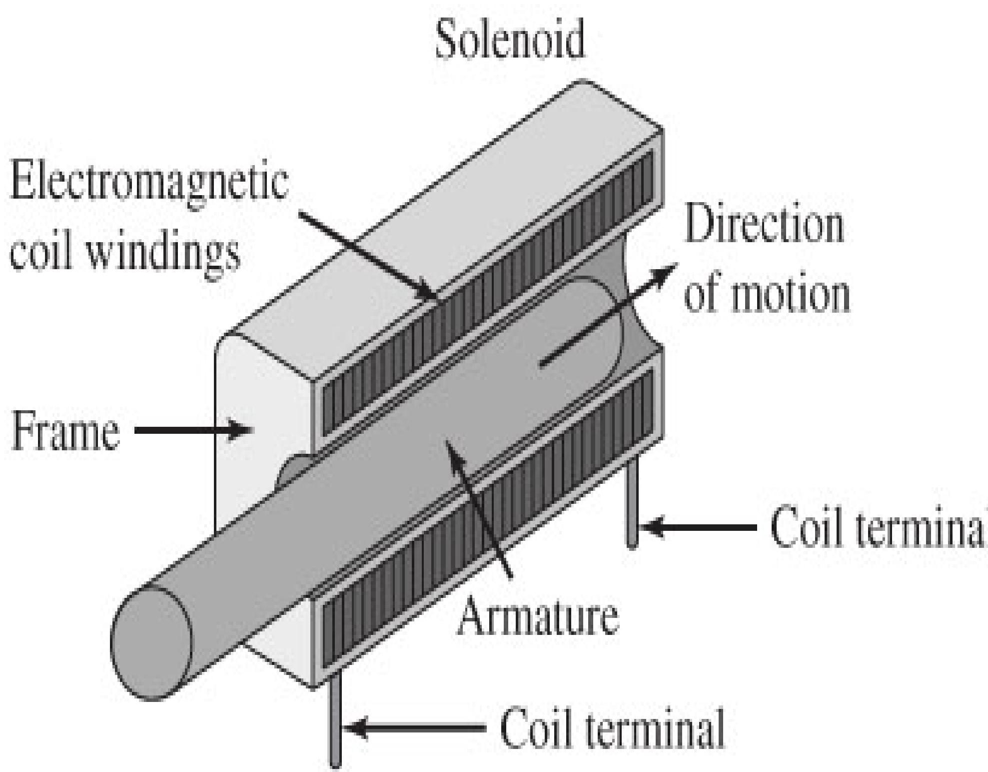
  - Quantization noise in timing…
    - Based on PWM clocks putting edge in the right place
    - Based on PDM having consistent clock lengths
  - Need enough bits in the PWM counter to manage cimting (8 bits or 16 bits)

- **If you are receiving PWM with a digital device need to do pulse capture**
  - Done using Pulse Accumulator hardware (or relevant software)
  - Can be used to measure frequency (time between edges)
  - Can be used to measure duty cycle (proportion of high to low times)
    - This is in Valvano, but not something we'll cover beyond this mention

Solonoids

- **Used to generate a short-stroke linear motion**
  - Release driven by spring, gravity, or second solonoid on same armature

Figure 8.64
Mechanical drawing of a solenoid showing that the EM coil causes the armature to move.

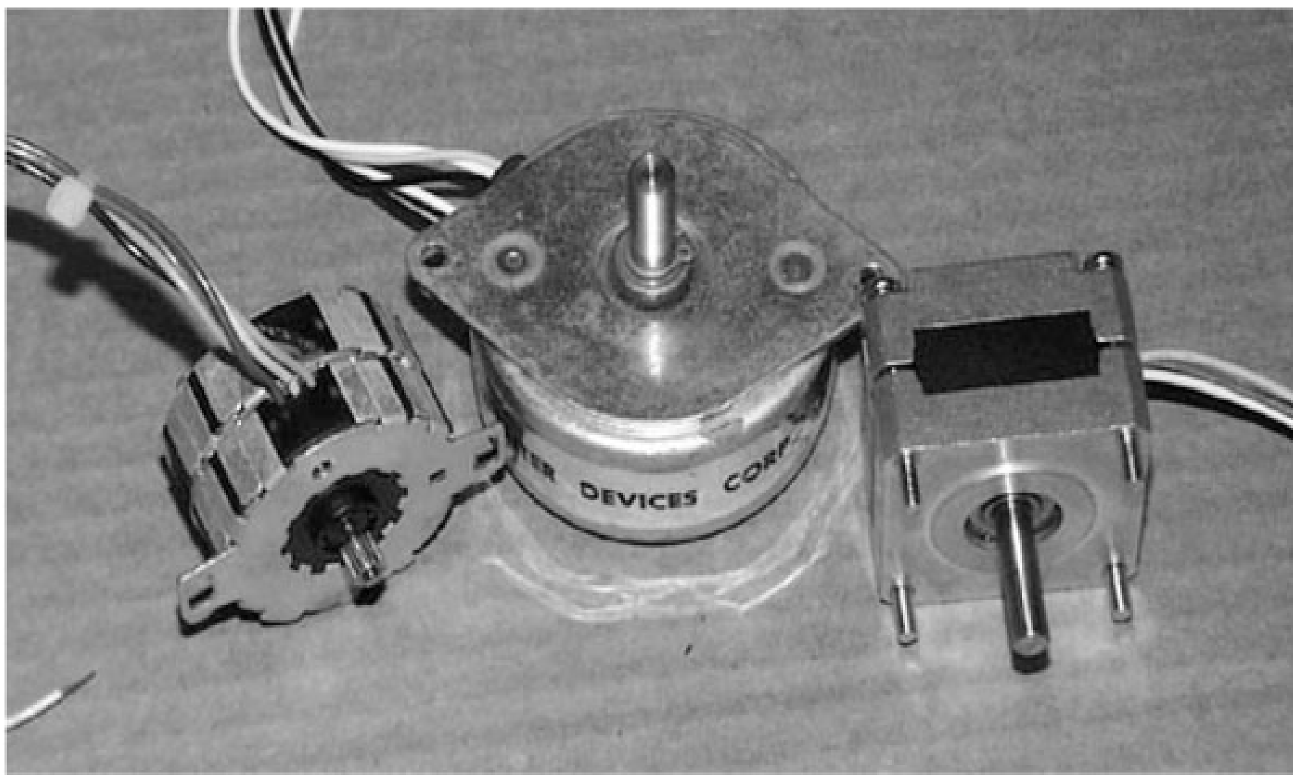


Stepper Motors

- **Many simple embedded systems use stepper motors**
  - Uses a digital (on/off) interface
  - Permits rotating motor to one of a set of rotational positions
  - Gives good positional stability without use of shaft encoder/feedback
  - General motor control is a whole other lecture (or set of lectures)

Figure 8.79
Three stepper motors.

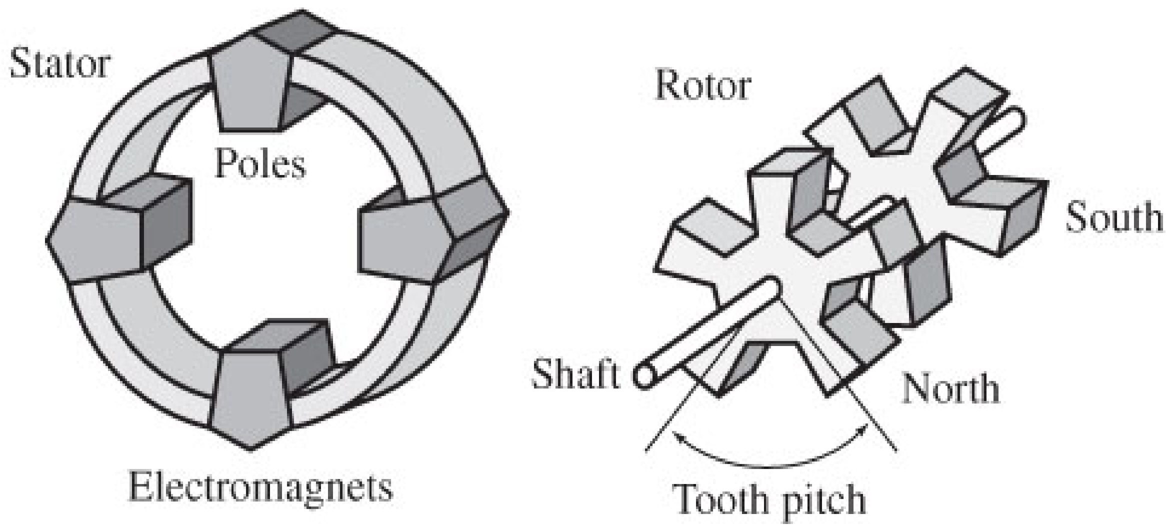
[Valvano]



Stepper Motor General Idea

- **Magnetic rotors spin, driven by electromagnetic stators**
  - Stator Poles alternate North and South to force motors to spin
  - (animation on following slides)

Figure 8.82
Simple stepper motor with 20 steps per revolution.



- – 20 steps per revolution ➔ 18 degrees per step
- – 200 steps per revolution ➔ 1.8 degrees per step
- – # steps per revolution = # stator coils (phases) * # teeth

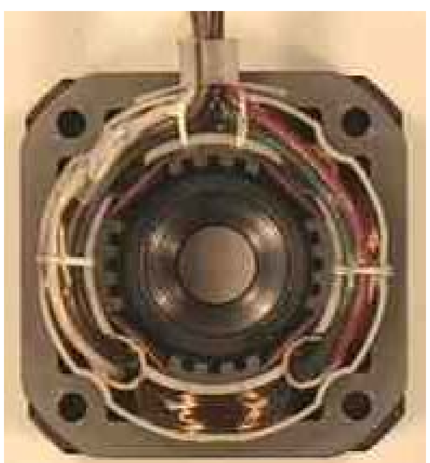
[Valvano]

Photos Of Stepper Motors

- http://www.doc.ic.ac.uk/~ih/doc/stepper/kp4m4/

KP4M4-001 Stepper Motor

+12v dc, four-phase, unipolar, permanent magnet, 3.6° per step



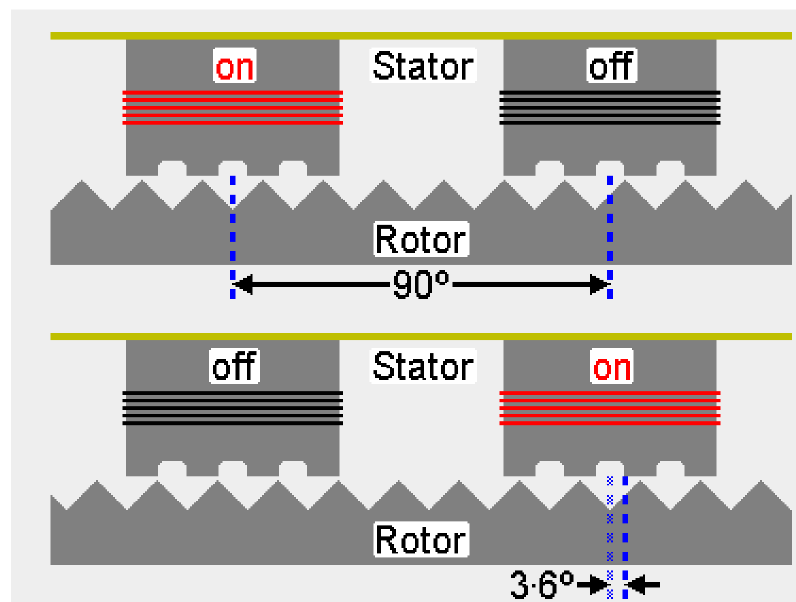
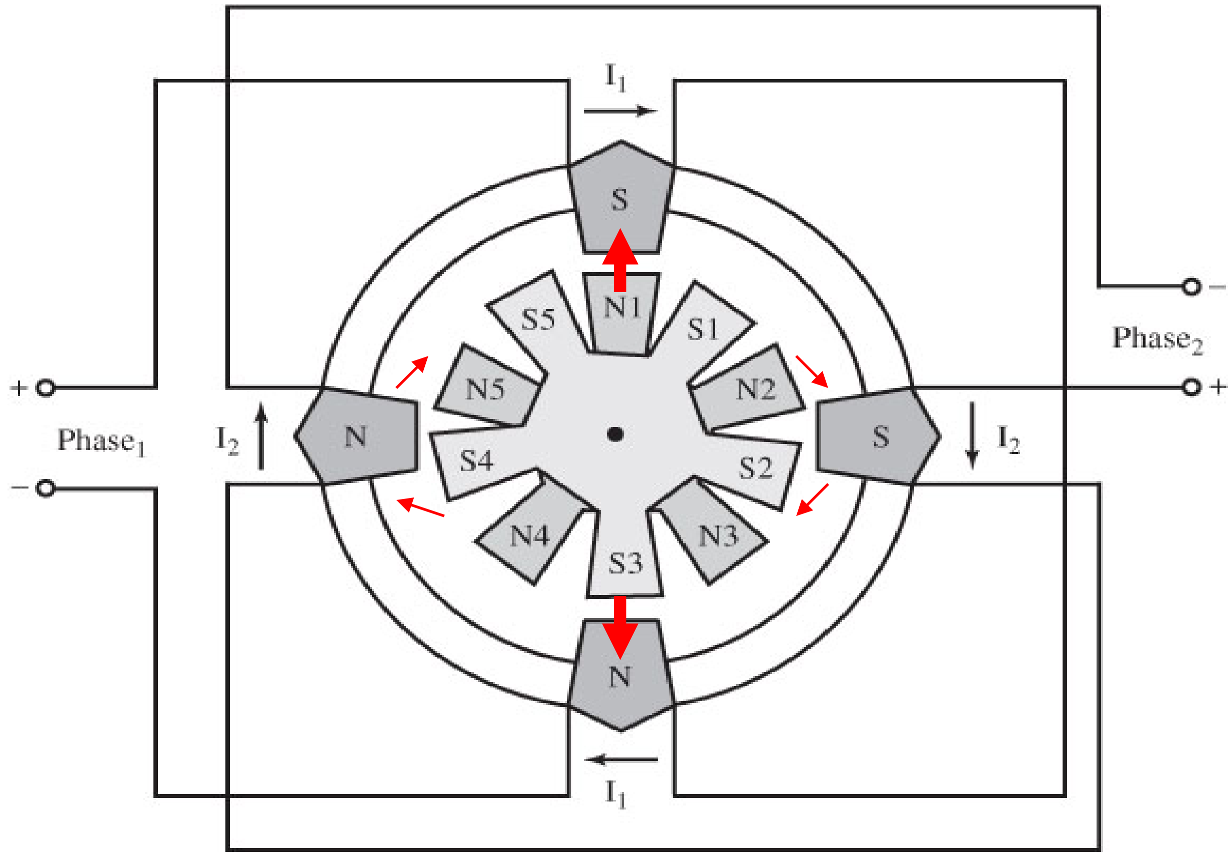
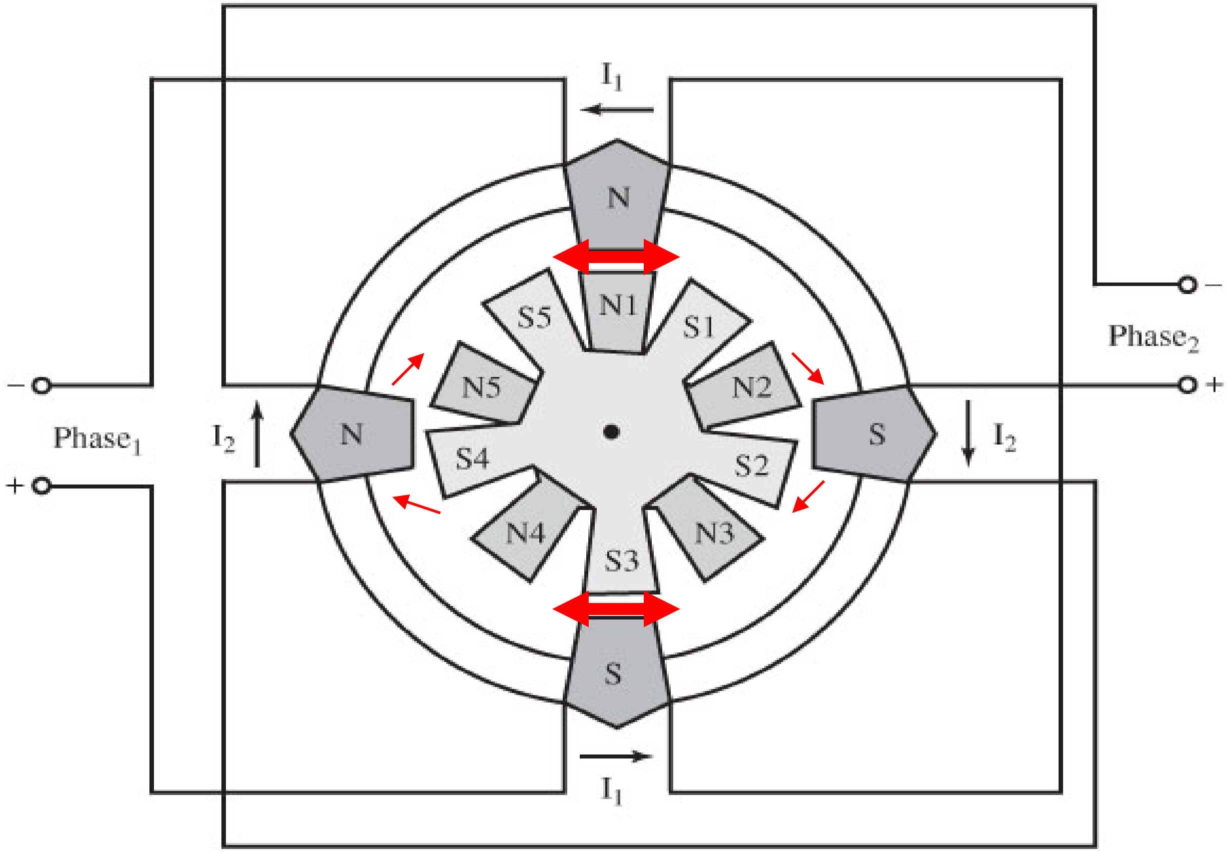


Figure 8.83
Stable state 1.



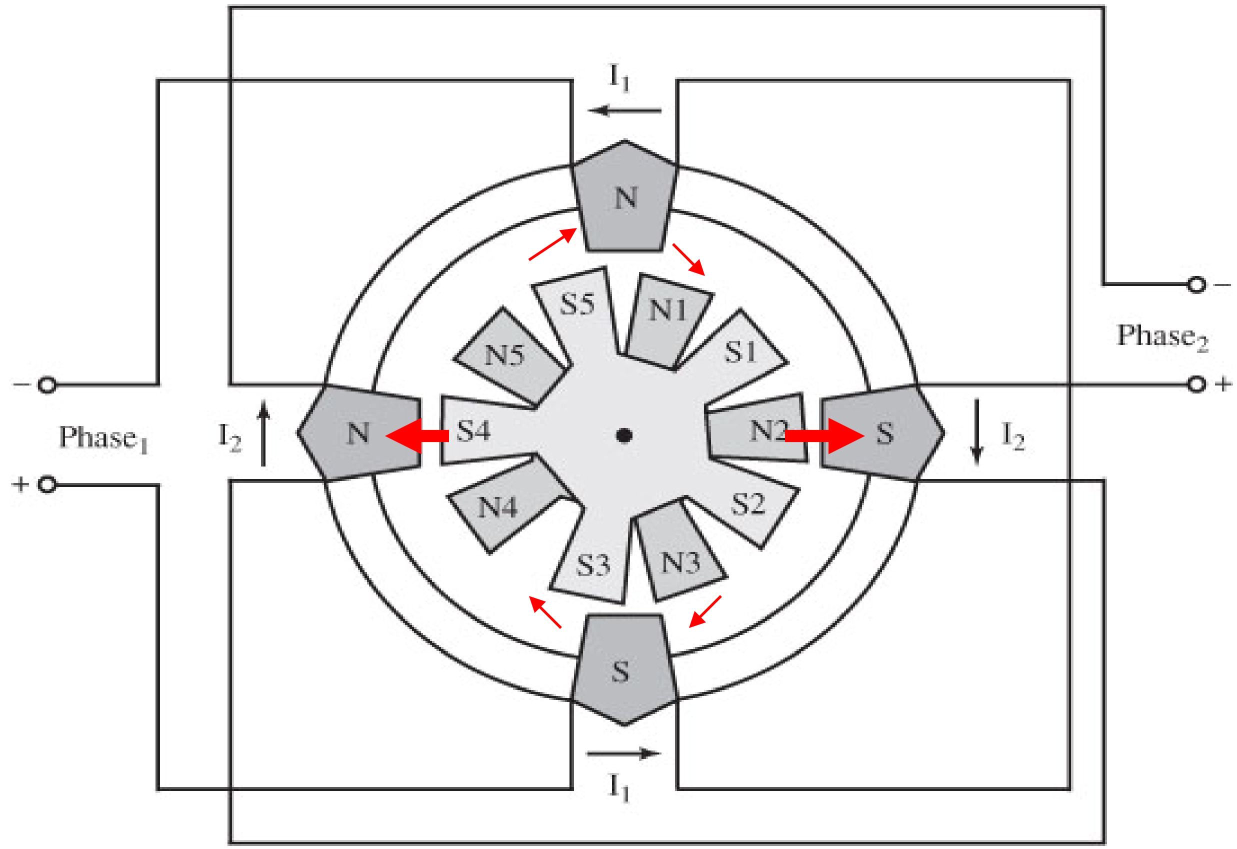
[Valvano]

Figure 8.84
Unstable state as rotor goes from state 1 to state 2.



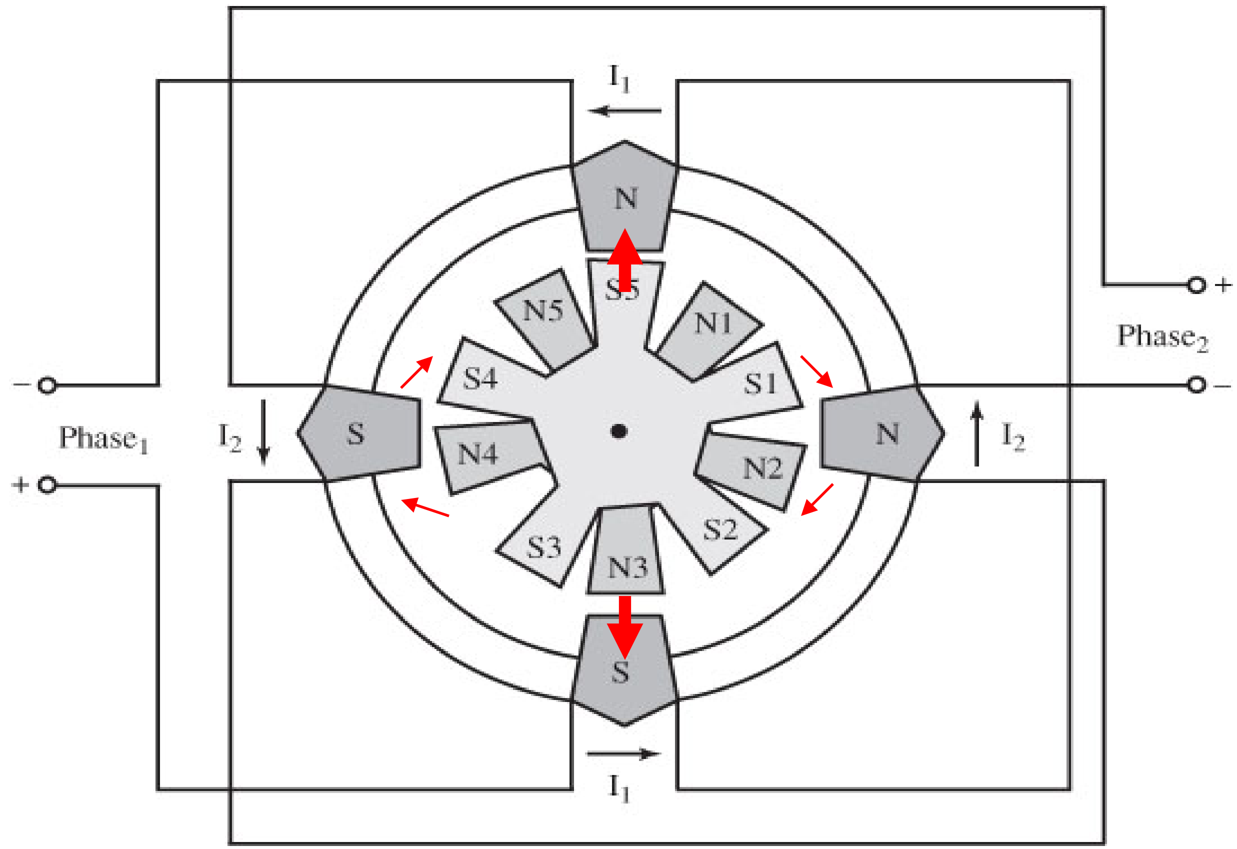
[Valvano]

Figure 8.85
Stable state 2.



[Valvano]

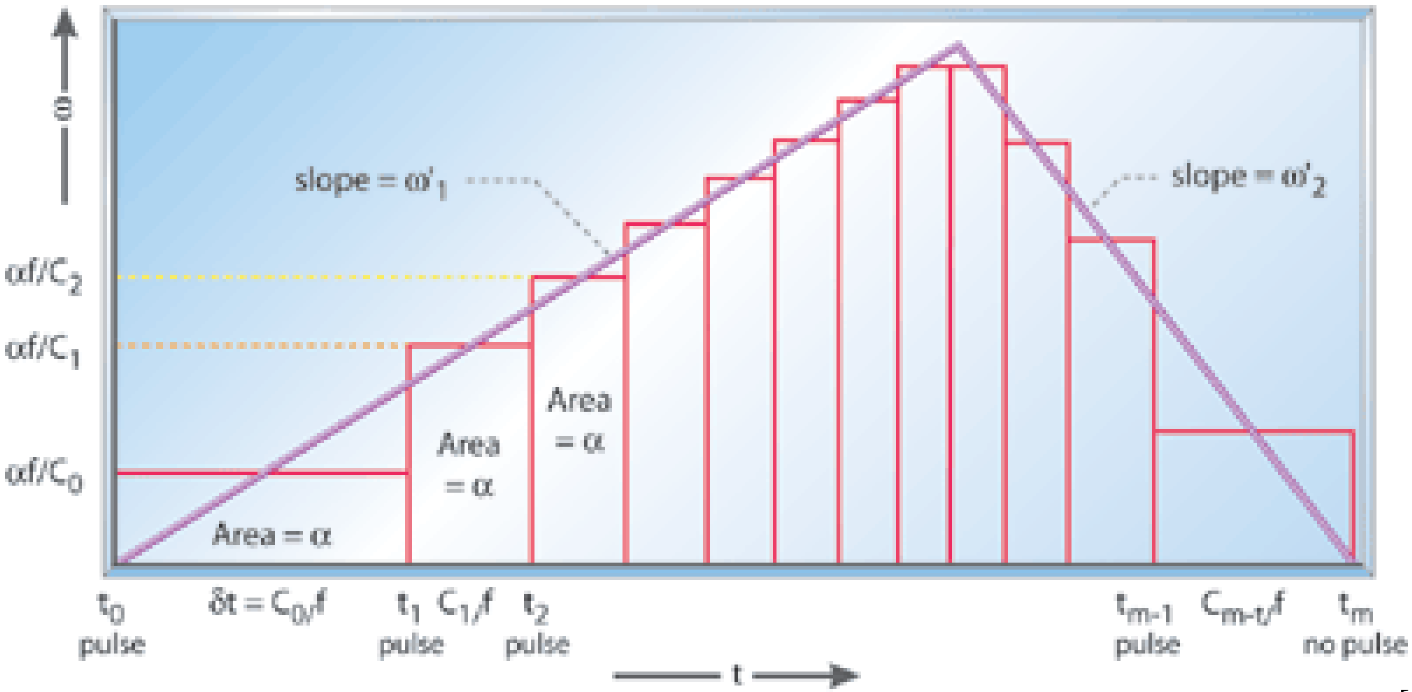
Figure 8.86
Stable state 3.



[Valvano]

Stepper Motor Ramp Up & Ramp Down

- **Stepper motor changes speed as it moves**
  - Magnetic pole changes have to coordinate with current speed
  - Motor spec & math gives you a speed profile



[Austin04]

Figure 1: Ramp geometry: move of $m = 12$ steps

- **What happens if you don't know where you are?**
  - Lose power
  - Controller resets
  - Something jams and you lose steps?

Stepper Motor Drive Circuit

- **Note: not the same motor type as other Valvano pictures**
  - A/A’ and B/B’ are always a high/low pair
  - High turns coil on; Low turns coil off
  - 1N914 diodes protect against back-EMF overvoltage when turning coil off

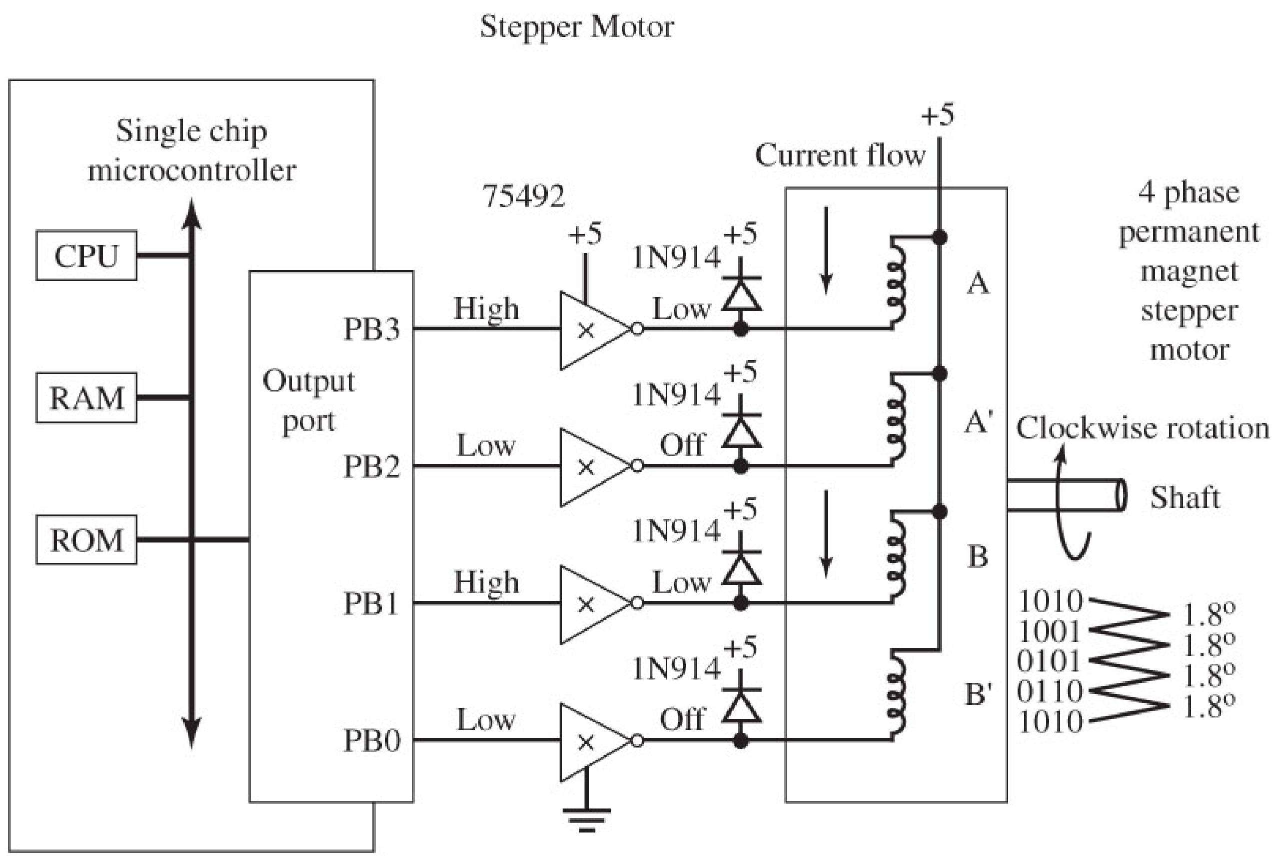
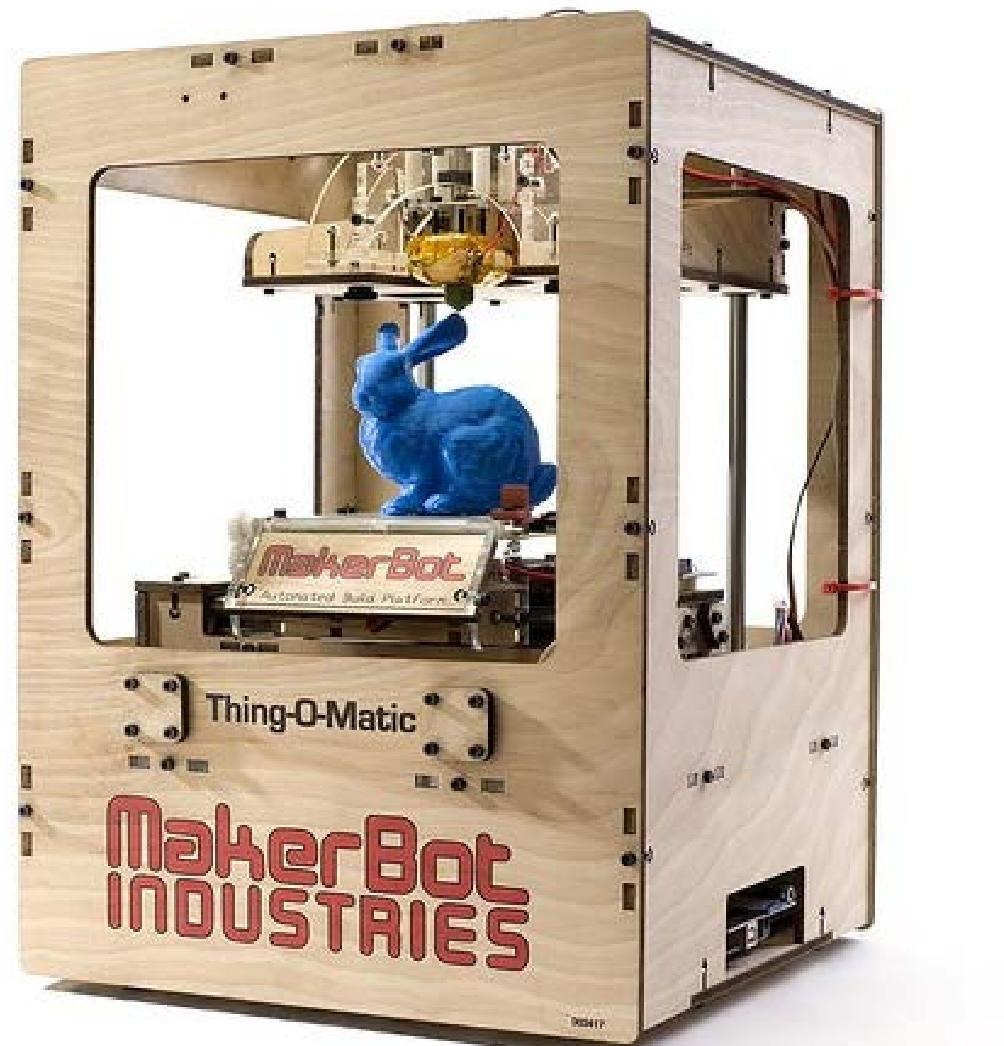


Figure 8.80
Simple stepper interface.

[Valvano]

Stepper Motors Are A Robot Gateway Device

- **Makerbot – stepper motors to position things**
  - The vibration from making "steps" makes noise. Can you do something fun with that?



http://store.makerbot.com/thing-o-matic-kit-mk7.html

Review

- **Digital To Analog Conversion**
  - Example implementation – how DAC actually works
  - Performance aspects: especially quantization issues
- **Encoding waveforms to feed to a DAC**
  - Low pass filter on outputs
- **Pulse Modulation**
  - Pulse Density Modulation vs. Pulse Width Modulation
  - How PWM works in general
  - For lab, be able to program the PWM hardware
  - How a servo works
- **Stepper motor**
  - Simplest kind of motor to use; have an idea of what’s going on with phases
  - And how a solonoid works