Von Neumann Architecture

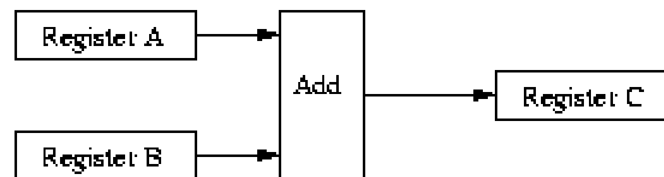
Combinational Circuits: output is a function of inputs.

State Machines: next state is a function of current state plus input; output output is a function of current state (Moore) or current state plus input (Mealy).

- Most real digital systems use some combination of the two.
- State machines, as finite automata, execute a deterministic process.
- They are limited in what they can compute (theoretically).

How can we make a more general/flexible/powerful digital system?

Consider a state machine that, on each clock cycle, takes in two numbers and outputs their sum on the next clock cycle.



- What if we use a general purpose ALU instead of an adder?
- Data comes into the system via registers A and B.
- Instructions enter the system via ALU function.

We know how to route data to different places (MUX)

- Why not add a bank of registers?
- Let the input data come from either the input or from the register bank.
- Let the output of the ALU go to any one of the register bank elements
- Note that we need more than one clock cycle to fetch, execute and store an operation.

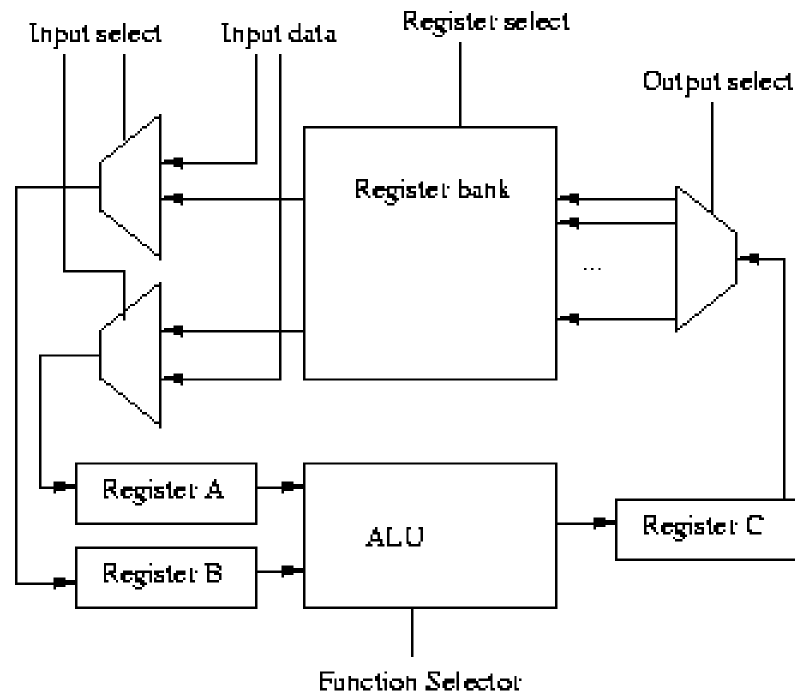
Why stop there?

- Let one or more of the registers be connected to an I/O port on the chip
- Let one or more of the registers control the input/output direction of a port
- **Memory-mapped I/O** lets input/output pins look like memory.

The big question is how we organize the series of operations the circuit will execute

- Could set the inputs by hand and clock the circuit.
- Would be better to have a memory with a series of operations stored in it.
- Could also have an external memory with lots of data in it.
- The series of operations to execute is a program.

©2006 Bruce A. Maxwell

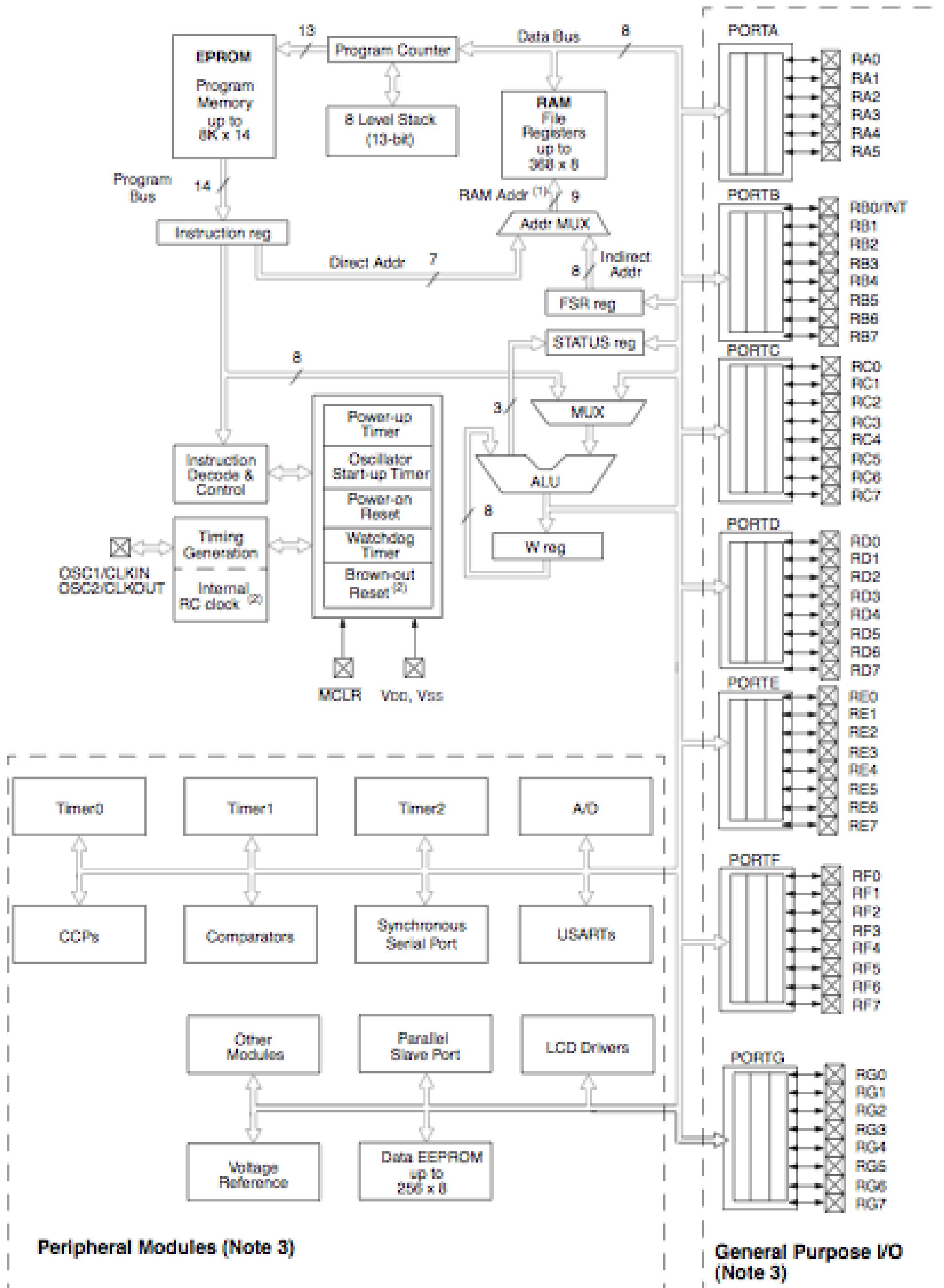


If we add external memory, how do we access it?

- We could let one of the registers be an address register for the external memory.
- Write the address to the address register, then route the data into Register A/B.
- We need to be able to put addresses into a program.
- We need a mechanism to get the address to the address register.

Fetch-Execute cycle

- The fetch-execute cycle begins with the fetch operation.
- The program counter specifies where to fetch the instruction.
  1. The PC address is applied to memory
  2. The instruction grabbed from memory is placed in the instruction register
  3. If there is an address with the instruction, it is placed in the memory address register [MAR].
- The data in the instruction register specifies where to grab the inputs.
- The data in the instruction register specifies what the ALU is to do.
- The data in the instruction register specifies where the output is to go.

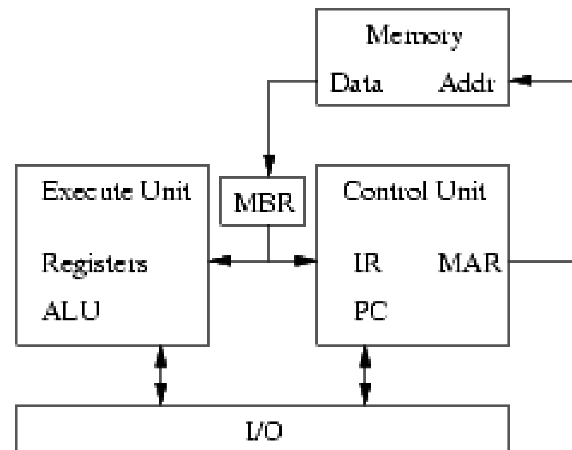


Note 1: The high order bits of the Direct Address for the RAM are from the STATUS register.
2: Not all devices have this feature, please refer to device data sheet.
3: Many of the general purpose I/O pins are multiplexed with one or more peripheral module functions. The multiplexing combinations are device dependent.

Taken from the PIC Mid-range Device Data Sheet

Von Neumann Architecture

The Von Neumann architecture consists of a control unit, execution unit, memory, and method of I/O.



The basic computer architecture follows a fetch-execute cycle.

- The fetch cycle is always identical: grab the next instruction from the program memory
- The execute cycle depends upon the instruction

The details of the execute cycle depend upon the architecture of the computer

- How many registers does it have?
- How much memory can it access?
- What functionality do the registers have?
- How many different operations can the computer execute?

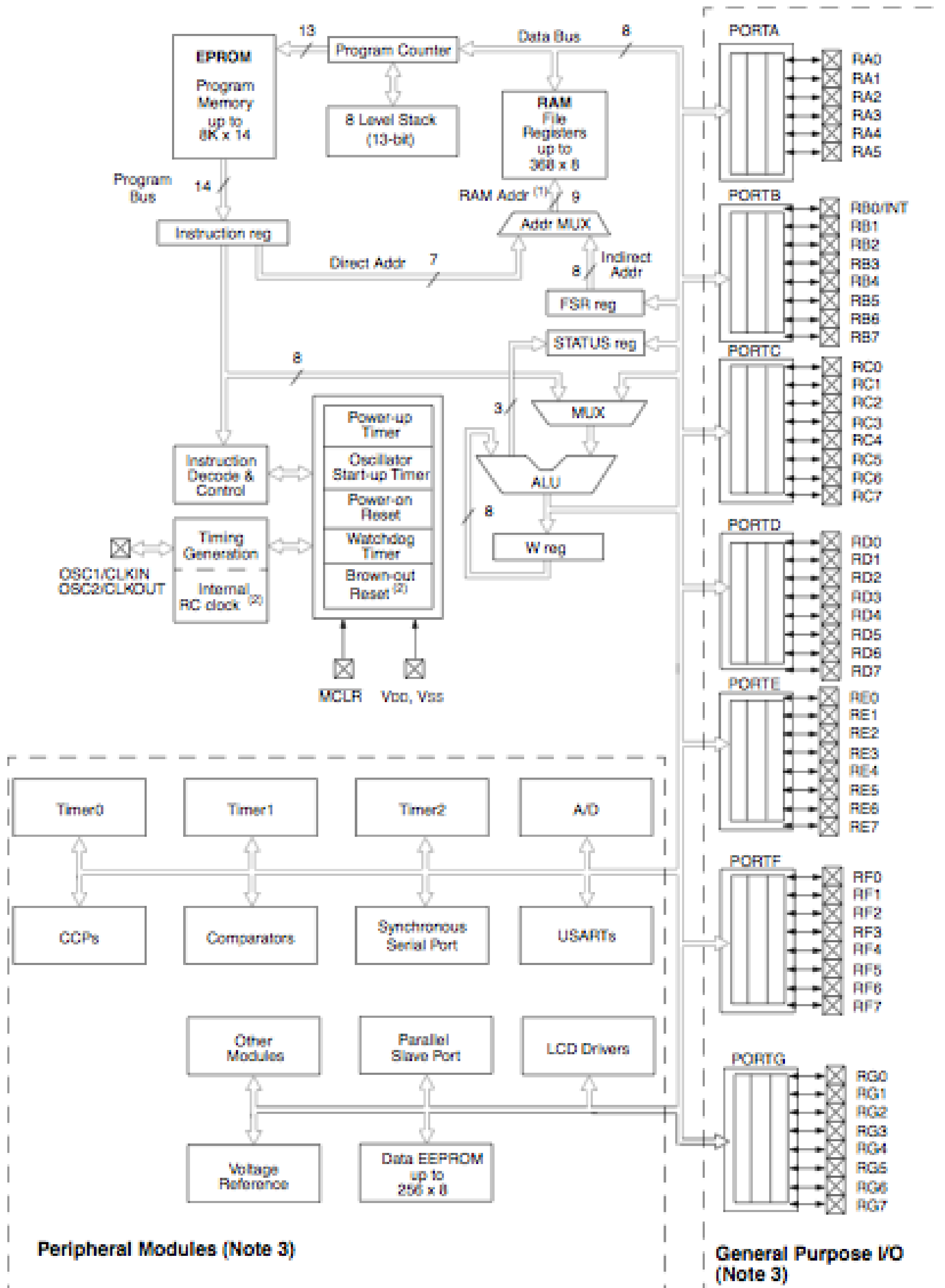
The architecture of the computer then determines the instruction set (iterative design process)

- The number of registers determines how many bits are needed to access each one.
- The functionality of the ALU determines how many bits are required to select an operation.
- The method of accessing memory determines how many address bits are required in an instruction.
- The need for constants within a program requires some bits be dedicated to operands

PIC Architecture

The PIC architecture is one example of a simple computer architecture

- There is a single working register, and all binary operations use it as one argument
- There are a limited number of instructions (only 35)
- The ALU, registers, and data memory are all designed for 8-bit values
- Each instruction is 14 bits, with 3 to 6 bits of opcode and the rest being data (address or value)



Note 1: The high order bits of the Direct Address for the RAM are from the STATUS register.
2: Not all devices have this feature, please refer to device data sheet.
3: Many of the general purpose I/O pins are multiplexed with one or more peripheral module functions. The multiplexing combinations are device dependent.

Taken from the PIC Mid-range Device Data Sheet

The Fetch process is identical for all instructions

1. The program counter value is applied to the EEPROM
2. The instruction register captures the next instruction
3. The program counter increments its value

The execute process depends upon the instruction. All binary operations (e.g. add, subtract, AND, OR, XOR) all follow the same execute process

1. One of the operands is the W register, which is always applied to one side of the ALU
2. One of the operands comes from either the instruction register or a file register
   - The address of the file register can be in the instruction (7 bits + 2 bits in status register)
   - The address of the file register can be in the FSR register (8 bits + 1 bit in status register)
   - For literal instructions (e.g. add 6 to W) the second operand comes from the IR
3. The operands get applied to the ALU
4. The output gets written to either the W register or the selected file register

All unary operations (e.g move, invert, increment, decrement) all follow the same process

1. The address of the operand is selected and applied to the file register
2. The source is routed to the ALU
3. The output of the ALU is routed to either the W register or back to the selected file register

To make the control unit for the PIC simple, all execute (and fetch) operations take 4 clock cycles. Sometimes operations do not need to use one or more of the cycles, so no operation occurs.

1. Q1: One clock cycle to propagate signals from the IR to the combinational circuitry
2. Q2: One clock cycle to propagate signals from the selected source to the ALU
3. Q3: One clock cycle to propagate signals through the ALU
4. Q4: One clock cycle to propagate signals from the ALU output to the destination

Absolute jumps (e.g. GOTO) are easy to implement on the PIC

1. The address to jump to is in the instruction (11 bits)
2. The address is applied to the ALU and passed through
3. The output is written to the PC

Conditional jumps are based on increment, decrement, or testing a bit. All conditional jumps only skip an instruction; they do not jump to a new place in the code.

1. The operand to increment, decrement, or test is moved through the ALU as usual
2. If the status flags indicate the skip condition is true, the next instruction is ignored
3. Processing then continues as usual, with no explicit modification of the PC

Interrupts

Unlike FPGAs, a CPU must execute its instructions serially. There is only a single ALU, and the pathways for moving data are limited. But concurrency is really nice because it lets you do multiple tasks simultaneously.

Sometimes, while running programs on a CPU, you want to having many things going on at once. For example, we may want the following processes going on simultaneously:

- Flashing an LED on a regular schedule
- Capturing an A/D value at a regular interval
- Executing a control algorithm that supervises both activities based on inputs

However, if we write a program to flash an LED that looks like:

```
while(1) {
  output_high(PIN_B3);
  delay_ms(200);
  output_low(PIN_B3);
  delay_ms(200);
}
```

Then the processor is doing nothing most of the time. This is not good.

While we have the constraint that the processor must execute serially, that does not mean that we have to do nothing during the delay time in the above loop. But we need a mechanism that lets us execute a different program while one program is waiting for something to happen. Alternatively, we can think about it as a mechanism that lets a second program take over the CPU for a short period of time, temporarily halting the execution of the first program. The mechanism used for this is an **interrupt**.

The basic idea is that when the CPU receives an interrupt signal, it follows a particular process to allow a second program to temporarily execute.

1. The CPU stores information about the current program (e.g. PC and status flags)
2. The CPU puts the address of the program to execute in the PC
3. The CPU starts executing at the new value of the PC
4. When finished, the CPU restores the PC and status flags
5. The CPU continues executing the original program

If the process is done correctly, the original program will not be aware anything has occurred (unless it happens to be checking a timer).

Interrupts

Unlike FPGAs, a CPU must execute its instructions serially. There is only a single ALU, and the pathways for moving data are limited. But concurrency is really nice because it lets you do multiple tasks simultaneously.

Sometimes, while running programs on a CPU, you want to having many things going on at once. For example, we may want the following processes going on simultaneously:

- Flashing an LED on a regular schedule
- Capturing an A/D value at a regular interval
- Executing a control algorithm that supervises both activities based on inputs

However, if we write a program to flash an LED that looks like:

```
while(1) {
  output_high(PIN_B3);
  delay_ms(200);
  output_low(PIN_B3);
  delay_ms(200);
}
```

Then the processor is doing nothing most of the time. This is not good.

While we have the constraint that the processor must execute serially, that does not mean that we have to do nothing during the delay time in the above loop. But we need a mechanism that lets us execute a different program while one program is waiting for something to happen. Alternatively, we can think about it as a mechanism that lets a second program take over the CPU for a short period of time, temporarily halting the execution of the first program. The mechanism used for this is an **interrupt**.

The basic idea is that when the CPU receives an interrupt signal, it follows a particular process to allow a second program to temporarily execute.

1. The CPU stores information about the current program (e.g. PC)
2. The CPU puts the address of the program to execute in the PC
3. The CPU disables other interrupts from taking place
4. The CPU starts executing at the new value of the PC
5. When finished, the CPU restores the PC
6. The CPU re-enables interrupts
7. The CPU continues executing the original program

If the process is done correctly, the original program will not be aware anything has occurred (unless it happens to be checking a timer).

The key to using interrupts successfully is that the interrupt handler (as such programs are called) must not take too much time. The original program may be watching a port, grabbing data, or otherwise doing processing that is time-critical. If the interrupt handler takes too much time, the original program may miss a deadline for action.

Interrupts are often used to monitor asynchronous inputs, since using the main program to poll ports is wasteful.

Since there can be multiple sources of interrupts, what program will execute when the CPU receives one? There are a number of possibilities.

- The CPU could always start the interrupt routine at the same memory location (e.g. 4)
- The CPU could look up the address of the appropriate interrupt routine by looking in a table indexed by the kind of interrupt that occurred.
- The CPU could receive the address of the interrupt routine from the device that produced the interrupt

If the CPU always starts the interrupt handler at a fixed address, then the first thing the interrupt handler must do is figure out what caused the interrupt so it knows which subroutine to execute.

Interrupts on the PIC microcontroller

The PIC has many interrupt sources: timers, external, peripheral devices, A/D converters, etc.

Interrupt Timing

1. External device, timer or peripheral generates an interrupt signal
2. Current instruction finishes executing, next instruction is fetched for the current program
3. PC is stored on the top of the system stack (dummy execution cycle)
4. 0x004 is loaded into the PC (dummy execution cycle)
5. Execution starts at the new PC value

The first thing the interrupt service routine [ISR] must do is store the **state of the computer**.

- The status flags need to be saved to memory
- The W register needs to be saved to memory
- Any other registers used by the ISR need to have their contents saved somewhere. Normally there will be a bank of registers that are set aside by all programs for this purpose (doesn't need to be a lot)

```
MOVWF  W_TEMP       move W to a temporary register (in current bank)
SWAPF  STATUS, W    swap the nibbles of the status register and put
                    the result in W (swap does not modify STATUS)
MOVEF  STATUS_TEMP  move W (which holds the reversed STATUS) to a
                    temporary register (in bank 0)
```

The second thing the ISR must do is **figure out which device caused the interrupt.**

- Each interrupt source has a flag associated with it in the Peripheral Interrupt Register [PIR].
- Each interrupt sources can be individually enabled or disabled using the Peripheral Interrupt Enable [PIE] register.
- The INTCON register contains flags and enable bits for CPU and external interrupts (e.g. overflow interrupts)
- The INTCON register also contains the Global Interrupt Enable [GIE] flag.

By polling the various bits in the INTCON and PIR registers the ISR can determine what device caused the interrupt. Most devices will have their own ISR subroutine that knows what to do for that device.

The ISR must not take too long to complete.

The last thing the ISR must do is **restore the W register and the status flags**.

```
SWAPF  STATUS_TEMP,W  swap the nibbles in STATUS_TEMP and put the
                      result in W
MOVWF  STATUS         move W to STATUS (now restored to its original
                      ordering)
SWAPF  W_TEMP,F       swap the nibbles in W_TEMP and return the
                      swapped value to it's original location
SWAPF  W_TEMP,W       swap the nibbles in W_TEMP and put the original
                      value of W back into W
```

How to use interrupts

So how do we use interrupts? For the most part, we use them either to watch for asynchronous inputs on peripherals or to handle tasks that need to be repeated on a regular interval (but for which we don't want to hang everything else).

If we are programming in C, the compiler takes care of the low level issues (like storing the W registers). Each interrupt will get its own function that will get called if the interrupt occurs. The things we still need to do are:

- Make sure interrupts are enable both globally and the specific one we need to watch.
- Make sure the ISR we write does not take long to execute
- Create global variables in the C code if we need the ISR to communicate with the main program

Sensing & A/D Interfaces

Digital circuits are very nice, because they have a natural buffer against noise

- It takes a lot of noise relative to the signal strength to send a zero to a one, or vice-versa
- We have straightforward digital means of detecting and correcting single bit-flips

But, the world is continuous

- How do we get from the analog world to the continuous world?
- We need to digitize/quantize/convert the continuous world to digital information
- We need to execute the quantization of the continuous world in such a way that we retain all of the important information.

What can we measure?

1. We can **count**: fingers, toes, ticks
2. We can measure **length**: we can hold up a reference stick
3. We can measure **voltage**: We can compare a voltage to a reference zener diode

Counting and voltages are the most useful for us to measure

- An oscillator can provide a time-varying signal that can drive a digital circuit (be converted to a clock)
- A voltage can be converted to a set of bits representing its value relative to a range
- We can measure distance by counting time of flight or counting rotations of a known radius circle

Keywords

- **Precision**: number of significant digits in the result
- **Accuracy**: how close you are to the true value
- **Sensitivity**: what is the smallest change you can sense reliably
- **Resolution (signal)**: # bits in an ADC
- **Resolution (time)**: sampling frequency
- **Resolution (spectral)**: how precisely can you measure frequency?
- **Dynamic Range**: range within which the sensor operates
- **Nyquist Criterion**: you have to sample at least twice the frequency you want to see

Remember: aliasing creates signals that don‘t actually exist

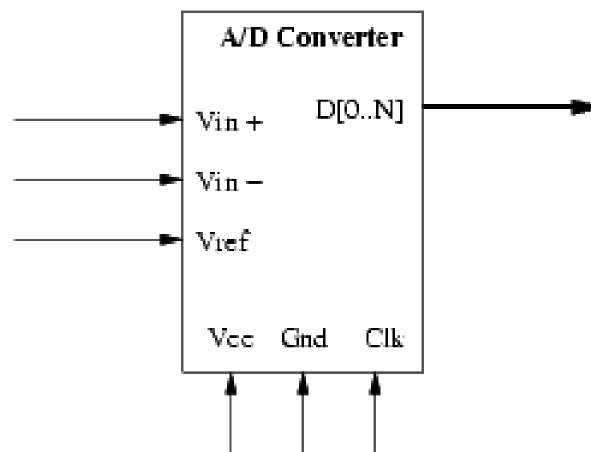
Measurement methodology

First you have to decide how you are going to measure something

- **Direct measurement**: directly comparing the measurand to a calibrated standard.
  - Meter stick
  - Zener diode
- **Indirect measurement**: measuring a quantity that is related to the thing you want to measure.
  - Measuring the wall of a furnace rather than sticking the thermometer inside
  - Measuring blood pressure using the Korotkoff sounds
- **Null Measurement**: comparing the measurand to a calibrated source and then adjusting one of them until the difference between them is zero.
  - Balance
  - Take a known reference voltage source connected to a potentiometer and an unknown voltage source and put them on a zero-center galvanometer (current flow meter). Adjust the pot until the galvanometer reads zero then read the pot.

A/D converters

A/D circuits are a common element of many real devices. They are the boundary between the continuous world and the digital world, and let us read sensors and other input devices constructed from analog circuitry.

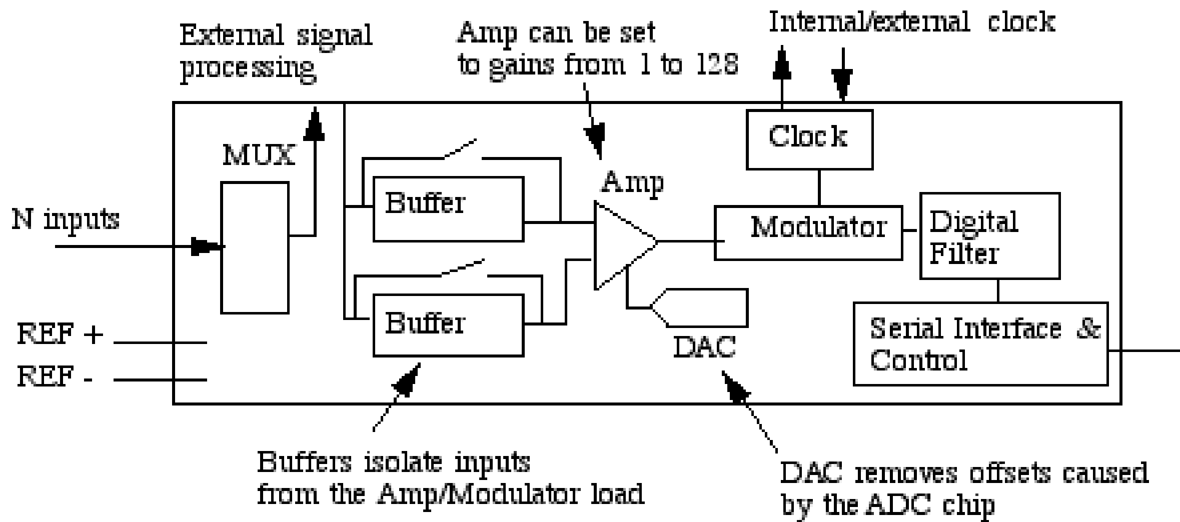


- One or more input pins, or pin pairs, that can handle voltages within a specific range
- N output pins that output the voltage level converted to an N bit number (usually binary)
- Analog reference voltage(s)
- Digital power and ground inputs (sometimes also used as reference voltage and ground)
- Enable or clock input pin(s)

Important characteristics of A/D converters

- **Digital Resolution**: number of output bits N per sample
  - The resolution of the A/D converter is defined by the number of bits per sample
  - N of 8 and 12 is common
  - N of 14, 16, 18 bits are now possible
  - In a 5V range, 8 bits = 20mV, 12 bits = 1.2mV, 16 bits = 80uV, 18 bits = 19uV (4800 sps)
- Number of **input channels** (independent signals)
  - Some A/D converters have an IN+ and IN- for each channel in addition to references
- Number of samples that can be take simultaneously, or **parallel inputs**
- **Parallel** or **Serial** output method
- Maximum **sampling frequency**
  - 1.5 Gsps is now possible at 8 bits: can pick up 750kHz signals (MAX108 8-bit A/D converter)
  - Allows direct RF/IF [Radio Freq/Intermediate Freq.] processing (don‘t have to do it in analog circuitry)
  - High-speed data acquisition
  - Digital Oscilloscopes
  - Radar/ECM systems

Design of an ADC: MAXIM MAX1400 ADC ($9)



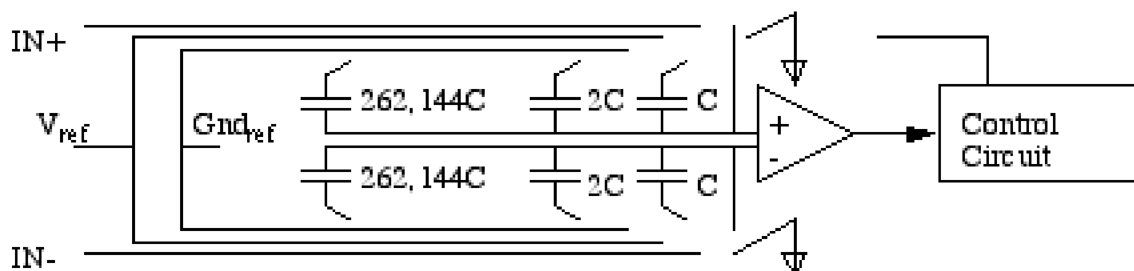
Resolution: 16 bits

Sample frequency: 480sps at 16-bits/sample, or 4800sps at 12-bits/sample

- This chip communicates via a serial interface (slow sampling rate)
- You can set it to scan all of the inputs sequentially and output the results
- The modulator (ADC component) and digital filter have multiple settings
- You can put external signal processing circuitry in between the analog MUX and the capture buffers that hold the signal values

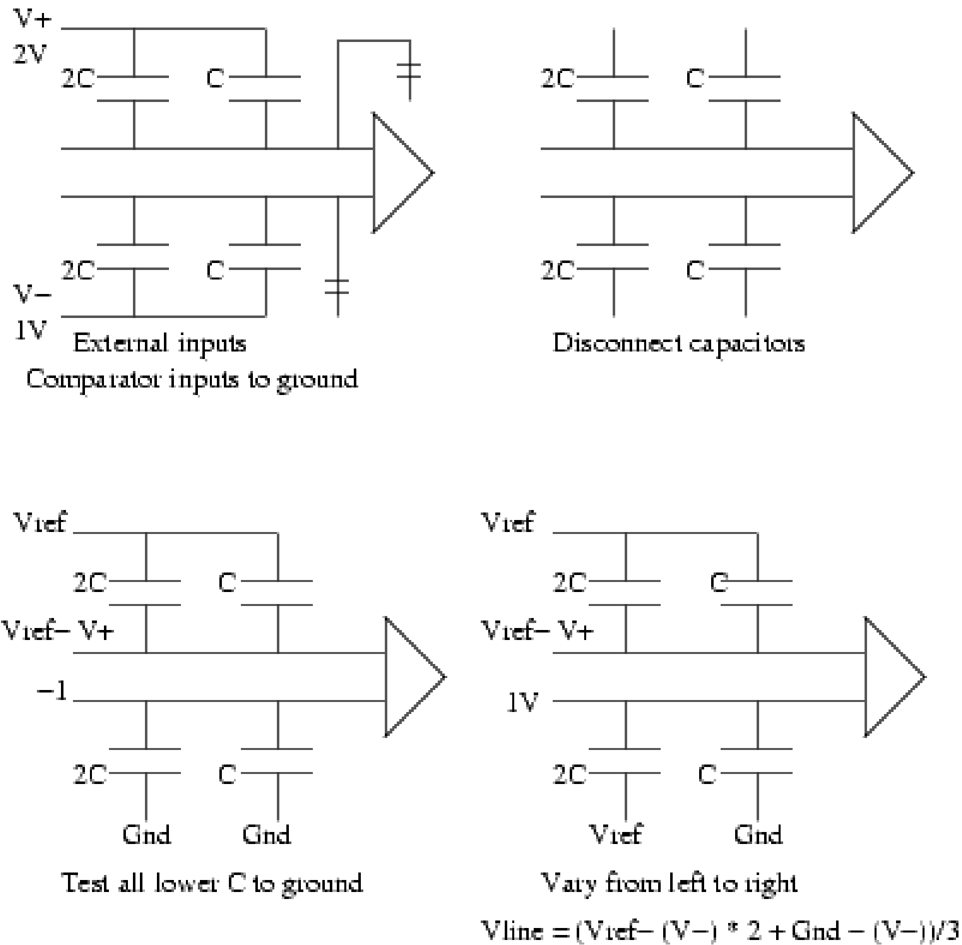
Analog Devices AD7674 PulSAR A/D Converter

- Differential input range of 5V
- Parallel and serial interface
- Feedback loop to match differential inputs by managing capacitor switches



ADC Process

- Comparator input lines are connected to ground, capacitors are connected to inputs
- Comparator input lines get disconnected from ground, capacitors disconnect from inputs
- Control circuit switches capacitors between Vref and Gndref until it achieves balance



Example: Initial charge on the capacitors is 2V on the top and 1V on the bottom, Vref is 3V

- Apply external voltages and set comparator inputs to ground
- Disconnect all the capacitors from the inputs
- Apply Vref to all the upper capacitors, top line goes to $V_{ref} - V^+$
- Apply ground to all of the lower capacitors
- Test bottom bits from left to right by first connecting to Gnd and then Vref
  - If the lower line is less than the upper line, flip the largest capacitor
  - Otherwise, hold the largest capacitor and move to the next one
- Voltage of the bottom line is a function of how the capacitors are connected

$$V_{lower} = \frac{(V_{ref} - V_{in}^-) \sum_{i \in V_{ref}} C_i + (Gnd - V_{in}^-) \sum_{j \in Gnd} C_j}{\sum_i C_i}$$

- Top/bottom = Vref/Vref is a 0, top/bottom = Vref/Gnd is a 1.
- Charge motion causes voltage changes; each smaller capacitor causes half as much change
- Central line connecting all capacitors together will always be a single voltage

Designing an A/D system

Decide on a measurement methodology: direct, indirect, or null measurement method

System Components

Transducer: converts one form of energy (information) to another (usually electrical)

- piezoelectric material: converts pressure to electricity
- silicon: temperature sensitive, EM sensitive, pressure sensitive
- spring: converts force to a distance, or possibly a resistance

Amplifier circuit:

- May include some processing before amplication (low/hi-pass filters)
- Amplifiers are low-pass circuits: higher the amplication the lower the cutoff frequency

Analog processing

- You can do quite a bit in analog hardware
- Can significantly reduce the load on a computer (microcontroller or PIC)
- Has to include a low-pass filter that meets the Nyquist criterion (¡ 2fs)
- May include a dc cutoff capacitor (hi-pass filter)

A/D converter

- Precision is determined by the number of bits
- Frequency is determined by the signal you want to measure
- Often needs to be much higher than just 2fs

Computer Port

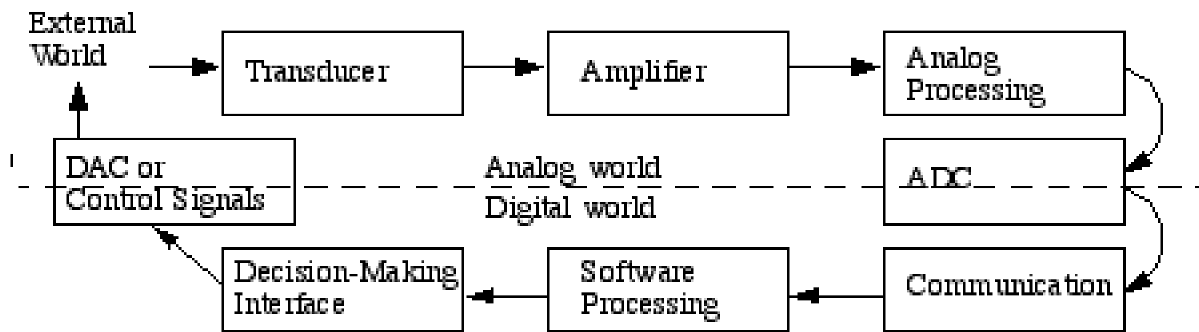
- Shielding issues
- Need a computer interface circuit
- Bandwidth issues

Software procesing

- Correctness
- Are you getting the right signal?
- Speed

Decision-making

- Control cycle, how do you make a decision once you have a measurement?
- What do you output to an operator or other computer system?


External
World
Transducer
Amplifier
Analog
Processing
DAC or
Control Signals
Analog world
Digital world
ADC
Decision-Making
Interface
Software
Processing
Communication

Designing an A/D system

Decide on a measurement methodology: direct, indirect, or null measurement method

System Components

Transducer: converts one form of energy (information) to another (usually electrical)

- piezoelectric material: converts pressure to electricity
- silicon: temperature sensitive, EM sensitive, pressure sensitive
- spring: converts force to a distance, or possibly a resistance

Amplifier circuit:

- May include some processing before amplication (low/hi-pass filters)
- Amplifiers are low-pass circuits: higher the amplication the lower the cutoff frequency

Analog processing

- You can do quite a bit in analog hardware
- Can significantly reduce the load on a computer (microcontroller or PIC)
- Has to include a low-pass filter that meets the Nyquist criterion (¡ 2fs)
- May include a dc cutoff capacitor (hi-pass filter)

A/D converter

- Precision is determined by the number of bits
- Frequency is determined by the signal you want to measure
- Often needs to be much higher than just 2fs

Computer Port

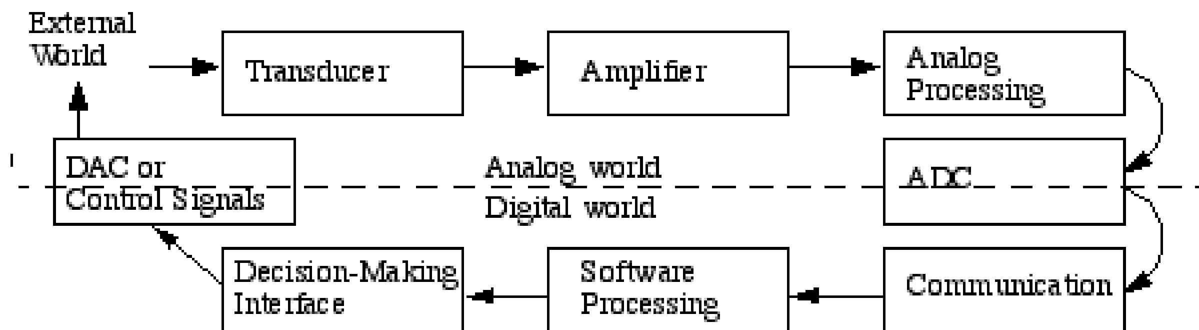
- Shielding issues
- Need a computer interface circuit
- Bandwidth issues

Software procesing

- Correctness
- Are you getting the right signal?
- Speed

Decision-making

- Control cycle, how do you make a decision once you have a measurement?
- What do you output to an operator or other computer system?



Sensing and Error

Types of measurement error

Theoretical error

- Transducers are pretty much all non-linear but you assume linearity within a range
- Equations used to produce the sensor output are approximations of the actual values
- Some sensors use a calibrated lookup table rather than equations (e.g. cheap cameras)

Static Error

- Interpolation errors: e.g., estimating distance between marks on a meter stick
- Environmental error: e.g., temperature sensitive circuits or measuring devices
- Manufacturing or design errors: e.g., offsets or flaws in the meter stick

Dynamic Errors

- The measurand might be moving while youre trying to measure it
- The inertia of mechanical devices may cause biases or offsets to measurements
- Measurements systems have frequency limitations
- Hysteresis in the measurement system: always approach a measurement from the same side

Instrument Insertion Error

- Schrödinger's cat (but at a much larger scale)
- The environment is altered by the measurement process

Nature imposed noise

Johnson Noise: thermodynamic origin

- A resistor has instantaneous current in it (which over time will create power)
- White noise phenomenon
- Constant in magnitude across the spectrum

Shot Noise: quantization of charge

- Also a white noise phenomenon

Flicker (1/f noise): origin is quantum mechanical

- Related to the uncertainty principle
- Increases in magnitude with decreasing frequency
- Engineers can always hang the blame on some particular device

These are nature imposed. They are lower bounds on how well you can do. It is always possible to do much worse.

Human imposed noise

Rotating machinery	Power Lines	Traffic
Radio	TV	Cell phones
Wi-fi	Microwaves	Radar

Other noise sources

Galactic effects	Continental drift	Seasons
Tides	Diurnal effects	Weather
Light	Heat	Cosmic rays

Consider, for example, how patterns of electrical, cell-phone, or wi-fi usage are influenced by the diurnal cycle and how that may impact the level of noise on a particular sensor system.

Solutions to measurement error

Cheap sensors are often cheap sensors.

- Cheap sensors are attractive, but to get the same quality signal as a more expensive sensor you may need lots of them.
- Cheap sensors may require more sophisticated designs surrounding them (which is often what you are getting when you purchase a more expensive sensor).

Use the instrument that least disturbs what is being measured to reduce instrument insertion error.

- Be aware of how your sensor interacts with the environment
- Using a sensor with an open conductive surface, for example, may not be wise if you are measuring electrical or magnetic fields.

Use several different instruments to measure the quantity

- Different sensors have different biases and noise sources
- If the biases and noise sources for the different sensors are independent of one another, then adding the results from multiple sensors will tend to cancel out the biases.
- Even different sensors that are of the same type may have different biases.

Use multiple measurements and average the results

- Will remove zero-mean Gaussian noise
- Will not remove salt & pepper noise
- Feasible for slowly moving signals

Use multiple measurements and take the median value

- Will remove salt & pepper noise
- Will probably be close to the true value unless the measurements are biased

Design stage solutions to noise reduction

Lock-in Amplifier

- Turn a slowly varying signal (but not the background) on and off at a relatively high but arbitrary frequency
- This creates an AC signal that is amplitude modulated by the information of interest
- Rectify the resulting signal and measure the magnitude
- Example: Measuring the light output of a candle in a lit room

Measuring differences directly: don‘t measure two large quantities and then subtract them, as this magnifies errors.

- Given Gaussian noise (noise defined by a mean μ and a standard deviation σ) the variances of two measurements add when you add or subtract the measurements.

$$\mu_{ab} = \mu_a - \mu_b \text{ and } \sigma_{ab}^2 = \sigma_a^2 + \sigma_b^2$$

Use noise to our advantage

- When digitizing a signal add some white noise
- Take multiple measurements of the signal
- The average will be more accurate when you add a small amount of noise
- What you are doing is equivalent to rounding to the nearest significant figure (LSB)

Watch out for antennas, magnetic pickup loops, and ground loops

- Like the big metal plate underneath the breadboards

Solutions to measurement error

Cheap sensors are often cheap sensors.

- Cheap sensors are attractive, but to get the same quality signal as a more expensive sensor you may need lots of them.
- Cheap sensors may require more sophisticated designs surrounding them (which is often what you are getting when you purchase a more expensive sensor).

Use the instrument that least disturbs what is being measured to reduce instrument insertion error.

- Be aware of how your sensor interacts with the environment
- Using a sensor with an open conductive surface, for example, may not be wise if you are measuring electrical or magnetic fields.

Use several different instruments to measure the quantity

- Different sensors have different biases and noise sources
- If the biases and noise sources for the different sensors are independent of one another, then adding the results from multiple sensors will tend to cancel out the biases.
- Even different sensors that are of the same type may have different biases.

Use multiple measurements and average the results

- Will remove zero-mean Gaussian noise
- Will not remove salt & pepper noise
- Feasible for slowly moving signals

Use multiple measurements and take the median value

- Will remove salt & pepper noise
- Will probably be close to the true value unless the measurements are biased

Design stage solutions to noise reduction

Lock-in Amplifier

- Turn a slowly varying signal (but not the background) on and off at a relatively high but arbitrary frequency
- This creates an AC signal that is amplitude modulated by the information of interest
- Rectify the resulting signal and measure the magnitude
- Example: Measuring the light output of a candle in a lit room

Measuring differences directly: don‘t measure two large quantities and then subtract them, as this magnifies errors.

- Given Gaussian noise (noise defined by a mean μ and a standard deviation σ) the variances of two measurements add when you add or subtract the measurements.

$$\mu_{ab} = \mu_a - \mu_b \text{ and } \sigma_{ab}^2 = \sigma_a^2 + \sigma_b^2$$

Use noise to our advantage

- When digitizing a signal add some white noise
- Take multiple measurements of the signal
- The average will be more accurate when you add a small amount of noise
- What you are doing is equivalent to rounding to the nearest significant figure (LSB)

Watch out for antennas, magnetic pickup loops, and ground loops

- Like the big metal plate underneath the breadboards

Numerical Methods: Differentiation

One of the most common operations in filtering, tracking, or control is to calculate the first derivative of a signal (rate of change). Differentiation is inherently a noise amplifying operation, especially for noise that is at a higher frequency than the signal.

Example: Derivative of $\cos(Kx) = -K\sin(Kx)$

When calculating deriatives numerically, you have the added problem of working in the digital domain with periodically spaced samples.

Its useful to look at the definition of a derivative.

$$\frac{dy}{dx} = \frac{\Delta y}{\Delta x} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} \tag{1}$$

When working with a data stream or digitized function, we have to approximate this as:

$$\frac{\Delta y}{\Delta x} = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \tag{2}$$

This is known as the forward difference function because it is based on the next sample.

If we want to approximate the true derivative, we need to make the step between x_{i+1} and x_i as small as possible. It turns out this is a really bad idea...

- You subtract two large numbers to get a much smaller number (lose information)
- You divide a large number by a small number (inherently unstable)
- The backward difference is no better, so increasing the sampling rate is not a total solution.

The **central difference** method uses both the forward and backward point to calculate the derivative at the central point.

$$\frac{\Delta y}{\Delta x} = \frac{f(x_{i+1}) - f(x_{i-1})}{x_{i+1} - x_{i-1}} \tag{3}$$

It turns out that the central difference will calculate better derivatives in general. It will also be more stable because the distance between x_{i+1} and x_{i-1} is twice that of x_{i+1} and x_i.

It also turns out that the central difference is derivable by using a Taylor series expansion of the function

$$f(x_{i+1}) = f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2!}h^2 + \dots \tag{4}$$

$$f(x_{i-1}) = f(x_i) - f'(x_i)h + \frac{f''(x_i)}{2!}h^2 + \dots \tag{5}$$

$$f(x_{i+1}) - f(x_{i-1}) = 2f'(x_i)h + \frac{2f'''(x_i)}{3!}h^3 \tag{6}$$

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} + O(h^2) \tag{7}$$

The interesting part of this derivation is to note that the error in the derivative is related to the square of the sampling period (the h^3 term is divided by h), which should be a small value.

Another way to think about the central difference formula is as a convolution of a smoothing function with the derivative function.

Consider, for example, the convolution of [1 1] and [-1 1]

- The result is [-1 0 1], which is the central difference theorem

It is possible to extend the concept to larger initial vectors like [1 2 1] and [-1 0 1]

- The result is [-1 -2 0 2 1], which is a smoother version of the central difference.
- Smoothing acts as a low-pass filter, reducing noise in the derivative signal

The derivative function you use for your application, which has to balance two factors:

- **Smoothness**: how noisy is the derivative signal relative to the underlying function
- **Responsiveness**: how quickly can the derivative change in response to the input

You can also calculate **higher accuracy derivatives** by again using the Taylor series: instead of lumping the second derivative in with the noise, solve for it. Solving for the second derivative pushes the noise into the cubic term.

If you substitute the result into the first derivative equation:

$$f'(x_i) = \frac{-f(x_{i+1}) + 8f(x_{i+1}) - 8f(x_{i-1}) + f(x_{i-2})}{12h} \tag{8}$$

Calculating **higher derivative** signals requires more data points, in general. You can recursively apply the forward, backward, or central difference formulas in order to make higher order derivative computations.

$$f''(x_i) = \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{\Delta x^2} \tag{9}$$

$$f'''(x_i) = \frac{f(x_{i+2} - 2f(x_{i+1}) + 2f(x_{i-1}) - f(x_{i-2})}{2\Delta x^3} \tag{10}$$

Just like the first derivative, these operators have their matrix equivalent

- Second derivative is the Mexican hat: [1 -2 1]
- Third derivative is the convolution of the first a second derivative operators

There are also higher accuracy derivative formulas for the higher derivatives

- In general, they require more data points
- In general, they will produce better estimates, but will be less responsive

A more expensive, but potentially smoother and more accurate method of calculating derivatives is to interpolate the points with an order N polynomial and then use the derivative of the polynomial. You need $N+1$ points for an order N polynomial, which will let you represent derivatives of order $N-1$.

- If you know your function is locally order N or less this can provide an accurate derivative.
- The Taylor expansion methods are actually based on a polynomial fit to the points
- Polynomial fitting is often used on large numbers of data points to get smooth derivatives

Higher dimensional derivatives

In higher dimensions (e.g. images), the easiest thing to do is to use 1D matrices to generate 2D matrices

- Take a 1D Gaussian [1 2 1]t and a 1D derivative [-1 0 1] and multiply them
- The result is an oriented derivative matrix (called the X Sobel in 2D)

$$\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{11}$$

- You can do the same for the Y direction by reversing the vectors
- The magnitude of a gradient in 2D is the sum of squares of the X and Y magnitudes
- The direction of the gradient in 2D is the atan(Y magnitude / X magnitude)

You can also get higher dimensional derivatives with the higher accuracy formulas: multiply a 1x5 Gaussian [1 2 4 2 1] and the higher accuracy derivative [-1 8 0 -8 1]

$$\begin{bmatrix} 1 \\ 2 \\ 4 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 8 & 0 & -8 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 8 & 0 & -8 & 1 \\ -2 & 16 & 0 & -16 & 2 \\ -4 & 32 & 0 & -32 & 4 \\ -2 & 16 & 0 & -16 & 2 \\ -1 & 8 & 0 & -8 & 1 \end{bmatrix} \tag{12}$$

Note that computation time in 2D goes up as N^2, where N is the size of the 1D operator.

Numerical Integration

Integration is another popular thing to do with incoming data

- Proportional-Integral control requires fast integration of an incoming stream
- Measurements of velocity or distance based on acceleration required accurate integration
- Pedometers, for example, that tell you how far youve walked

The simple methods for integrating a data series are really simple. These all fall under the rubric of Newton-Cotes formulas, where we approximate the function $f(x)$ with an easily integrable function. The error is related to the quality of the approximation.

Rectangular rule: approximate the function as a series of rectangles and sum their area

$$I = \sum_N f(x_i)\Delta x \tag{1}$$

Trapezoidal rule: approximate the function as a series of trapezoids and sum their area

$$I = \sum_N \frac{f(x_i) + f(x_{i+1})}{2}\Delta x \tag{2}$$

Simpsons 1/3 rule: approximate the integral using a series of parabolas (2nd order curves)

$$I = \sum_N \left[f(x_{i-1}) + 4f(x_i) + f(x_{i+1})\right]\frac{1}{3}\Delta x \tag{3}$$

Simpsons 3/8 rule: approximate the integral use a cubic polynomial

$$I = \sum_N \left[f(x_i - 1) + 3f(x_i) + 3f(x_{i+1}) + f(x_{i+2})\right]\frac{3}{8}\Delta x \tag{4}$$

If you work through the Taylor series...

For the rectangular rule, the error is proportion to the time step, $O(\Delta x)$

$$E = \int_{x_i}^{x_{i+1}} f(x)dx - f(x_i)\Delta x \tag{5}$$

$$f(x) = f(x_i) + (x - x_i)f'(x_i) + f''(x_i)\frac{(x - x_i)^2}{2!} + \dots \tag{6}$$

$$\int_{x_i}^{x_{i+1}} f(x)dx = \int_0^{\Delta x} \left(f(x_i) + yf'(x_i) + \frac{y^2}{2!}f''(x_i) + \dots\right) dy \tag{7}$$

$$E = \int_{x_i}^{x_{i+1}} f(x)dx - f(x_i)\Delta x = \left[f(x_i)\Delta x + f'(x_i)\frac{\Delta x^2}{2} + f''(x_i)\frac{\Delta x^3}{6} + \dots\right] - f(x_i)\Delta x \tag{8}$$

Thus, the error per step is proportional to , which is the largest remaining term.

However, if we have N segments, then we need to sum the error over the segments

$$E \approx \sum_N \frac{\Delta x^2}{2} f'(x_i) = \frac{\Delta x^2}{2} \bar{f}' N = \frac{(b-a)^2}{2N^2} \bar{f}' N = \frac{b-a}{2} \Delta x \bar{f}' \tag{9}$$

The range [a, b] is the overall range of the integral, and $\frac{(b-a)}{N}$ is equal to Δx.

- For the rectangle rule, the error is $O(\Delta x)$
- For the trapezoidal rule, the error is $O(\Delta x^2)$
- For both the 1/3 and 3/8 rule the error is $O(\Delta x^4)$

Reducing errors

In order to reduce integration errors you either need to decrease the step size or increase the order of the integration formula

- Decreasing step size at some point starts to increase round-off and numerical errors
- Increasing the order of the integation increases the computational load

It turns out you can use the results of lower order integrations to calculate more accurate results

Richardson extrapolation

If you take two estimates of the integral, but use different step sizes, then you can write an approximation to the error term in terms of the two results. For the trapezoidal rule, the expressions for the integral using the two step sizes are:

$$I \approx I_1(\Delta x_1) + c\Delta x_1^2 \tag{10}$$

$$I \approx I_2(\Delta x_2) + c\Delta x_2^2 \tag{11}$$

These just say that we have two estimates of the true integral I. One is based on time step Δx_1, and one is based on time step Δx_2. Subtracting these two equations gives an estimate for c, which is a multiplier for the error term, in terms of the integral results and the two step sizes.

$$c \approx \frac{I_2(\Delta x_2) - I_1(\Delta x_1)}{\Delta x_1^2 - \Delta x_2^2} \tag{12}$$

If we then solve for I by substituting back into (11), we get an improved estimate of I.

$$I \approx I_2(\Delta x_2) + \frac{I_2(\Delta x_2) - I_1(\Delta x_1)}{\left[\left(\frac{\Delta x_1}{\Delta x_2} \right)^2 - 1 \right]} \tag{13}$$

Now if we simplify things by using $\Delta x_2 = \frac{1}{2}\Delta x_1$ then we actually get Simpsons 1/3 rule with step size Δx_2. This means that using a weighted sum of two trapezoidal rule integral results gives us the equivalent of a 2nd order polynomial result and an error of $O(\Delta x^4)$.

Romberg Integration

The end result is that you can define a recursive procedure that calculates increasingly more accurate integrals based on trapezoidal rule results, which are fast to compute.

The general formula for developing higher accuracy results is:

$$I_{k,n} = \frac{4^k I_{k-1,n} - I_{k-1,\frac{n}{2}}}{4^k - 1} \tag{14}$$

k is the order of the integration, and n is the number of samples integrated by the process. A simple trapezoidal rule (order 0) integration result using n steps would be represented as $I_{0,n}$.

Given two trapezoidal rule results $I_{0,n}$ and $I_{0,\frac{n}{2}}$ with step sizes of $\frac{\Delta x}{2}$ and Δx, respectively, you can combine them to get the Simpson's rule (1st order) result using the following.

$$I_{1,n} = \frac{4 I_{0,n} - I_{0,\frac{n}{2}}}{4^1 - 1} \tag{15}$$

The formula can be used recursively to generate increasingly accurate estimations of the integral in a tree structure. Each step up the tree results in an accuracy increase on the order of Δx^2.

For example, to calculate a second order result for 16 samples with an error on the order of $O(\Delta x^6)$, represented as $I_{3,16}$, we would make the following calculations.

1. For the order zero level, we would calculate $I_{0,4}$, $I_{0,8}$ and $I_{0,16}$ using the trapezoidal rule. For $I_{0,4}$ we would use four evenly spaced samples out of the original 16 (e.g., 0, 5, 10, and 15). For $I_{0,8}$ we would use eight samples (e.g., 0, 2, 4, 6, 8, 10, 12, and 14), and for $I_{0,16}$ all 16 samples. These would be the only calculations using raw input values.
2. For the order one results, we would calculate $I_{1,8}$, and $I_{1,16}$ using the recursive formula given in (14).
3. For the final order two result, we would calculate $I_{2,16}$ using the recursive formula in (14).

The end result is a more accurate estimate of the true integral while only computing trapezoid rule integrals. Trapezoid rule integrals are easy to do on a microprocessor such as the PIC because they only involve divisions by two, which can be accomplished by a bit shift.

General models of communication

Strobing

One unit controls the interaction

Sending Process

- Strobe active indicates unit has information to send
- Strobe and data go inactive after some period of time
- Has to wait as long as the slowest device before removing the data

Receiving information

- Strobe active indicates unit wants some information
- Data is read after some time period
- Strobe goes inactive after the data has been read
- Has to wait as long as the slowest device before reading the data

Problem: you have no idea what the other device is doing

Handshaking

Both units communicate their readiness

Sending information

- Data goes on the data lines
- Sending unit sets strobe/request active to indicate the data is ready
- Receiving unit sets reply high to acknowledge it has received the request
- Sending unit sets strobe/request inactive to indicate it is about to remove the data
- Receiving unit sets reply inactive to indicate it is ready for another transmission

Receiving informaiton

- Receiving unit sets strobe/request active to indicate it wants some data
- Sending unit sets reply active after it has put the data on the data lines
- Receving unit sets strobe/request inactive after it has read the data
- Sending unit sets reply inactive when it is ready to receive another request

Strobing: Parallel Port Communication

Centronics mode (how data used to be sent to printers)

One-way communication from host to the peripheral controlled by the host

- Uses the 8 data lines and a strobe
- Peripheral has a busy line and an ack line, but they may not be acknowledged by host

Process

1. Host places data on the data lines
2. Host checks if busy line is low (peripheral ready?)
3. Host asserts nStrobe (active low)
4. Host de-asserts nStrobe

The peripheral may assert busy/ack lines while reading the data. Usually, the peripheral uses the rising edge of nStrobe to clock the data into a buffer register on the peripheral.

Serial Communication

Asynchronous communication protocol

1. Start bit (0) (opposite of idle line)
2. Character bits (7 or 8 of them)
3. Parity bit (1)
4. Stop bit (1-2 of them) (equal to idle line)

Synchronous transmission does not use the start and stop bits, but must use timing signals to keep the bits synchronized to the clock.

For asynchronous communication, the speed at which bits are sent is fixed through negotiation at startup, or simply by a fixed setting. Hence, a clock that is 3-4 times faster than that is sufficient to guarantee that the circuit is sampling the bits properly.

Example asychronous receiver

Receiver is based on a fast clock (e.g. 4x faster than the transmission rate)

- Process initiates on a start bit assertion, sample time is set to be in the middle of a bit
- Character bits get read in the middle
- Parity bit is read
- Receiver indicates to host that data is ready and resets back to the idle state

Digital Signal Processing

Common techniques in digital signal processing include

- Averaging samples across time to reduce noise
  1. Box filter: weight each sample in time equally and average
  2. Gaussian filter: weight each sample in time according to a Gaussian distribution

  The method of averaging, and the time span used to generate the average, affect the characteristics of the measurement and the responsiveness of the system to change.
- Low-pass filtering: pick a cutoff frequency and design a digital filter for it (Z-transform design space). Averaging is a form of low-pass filtering.
- High-pass filtering: pick a cutoff frequency and design a digital filter for it (Z-transform design space). Taking derivatives is a form of high-pass filtering.

  Both low-pass and high-pass filters can also be executed in the Fourier domain using the discrete Fourier transform, or the fast Fourier transform for sample sizes that are a power of 2. Generally, a Hamming window is used to create the time slice used to generate the Fourier transform.
- State estimation: each measurement is a function of the state of the system with added noise. How can we optimally estimate the system state over time given A) a model for the relationship between the measurement and the system state and B) a certain level of noise?

Kalman Filtering

Kalman filtering is optimal state estimation from noisy measurements

- All measurements include some noise, sometimes the process itself is noisy, too
- Each measurement contains some information about the state
- In addition, you may know something about how the state is expected to change
- The better your measurement accuracy, the more you trust the measurements
- The more noise that exists in the system, the more you want to average over time

State can be anything you are trying to measure, including things you measure indirectly

- Temperature: best estimate of a single parameter
- Location: (x, y, θ) robot localization using GPS navigation systems
- Trajectory: $(x, y, z, \dot{x}, \dot{y}, \dot{z})$ radar and missile tracking

The Kalman filter is an optimal algorithm for estimating state given the following conditions

- The system is linear (describable as a system of linear equations)
- The noise in the system has a Gaussian distribution
- The error criteria is expressed as a quadratic equation (e.g. sum-squared error)

Example of a linear system: Basic Newtonian physics

$$\begin{bmatrix} x_{i+1} \\ \dot{x}_{i+1} \\ \ddot{x}_{i+1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t & \frac{1}{2}\Delta t^2 \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ \dot{x}_i \\ \ddot{x}_i \end{bmatrix} \tag{1}$$

Setting up the Kalman filter

1. A measurement z_k is linearly related to a system state x_k as in (2), where the noise term v_k is a Gaussian (normal distribution) with zero mean and covariance R_k.

$$z_k = Hx_k + v_k \tag{2}$$

2. If variations in x and z are Gaussian (normally distributed), then the optimal estimator for the next state $\hat{x}_k(+)$ is also the optimal linear estimator, and the estimate of the next state is a linear combination of an a priori state estimate $\hat{x}_k(-)$ and the measurement z_k.

$$\hat{x}_k(+) = A\hat{x}_k(-) + Bz_k \tag{3}$$

3. The system to be measured is represented by a system dynamic model that says that the next state is a linear function of the current state plus some noise w_{k-1}. Note that, in the absense of any measurements, the error in the system state estimation will grow without bounds at a rate determined by Q_k, the process covariance matrix.

$$x_k = \Phi_{k-1}x_{k-1} + w_{k-1} \tag{4}$$

4. The system model, which we create for a specific system using our knowledge of its physics, tells us the a priori estimate of the next state (the estimate prior to any measurement information) based on the last a posteriori state estimate.

$$\hat{x}_k(-) = \Phi_{k-1}\hat{x}_{k-1}(+) \tag{5}$$

5. We also have an error model that tells us how much error currently exists in the system. The new covariances of the error are linear functions of the old error and the system update transformations in (5). The new covariance also increments by the process covariance matrix (the inherent error in the state update equation (4)).

$$P_k(-) = \Phi_{k-1}P_{k-1}(+)\Phi_{k-1}^T + Q_{k-1} \tag{6}$$

6. Now we have estimates of the new a priori state and a priori error based on the state and error update equations. These are open loop equations that do not use any new measurement information. Now we need to update the state based upon the measurements, which means we need to estimate the Kalman gain matrices for (3). The Kalman gain needs to balance how much to trust the state update (using $P_k(-)$) and how much to trust the measurement (using R_k). The Kalman gain matrix equation is given in (7). The expression is the process noise projected into the measurement domain, so the Kalman gain matrix is the process covariance divided by the process covariance plus the measurement

noise. The lower the measurement error relative to the process error, the higher the Kalman gain will be.

$$\bar{K}_k = P_k(-1)H_k^T \left[H_k P_k(-)H_k^T + R_k\right]^{-1} \tag{7}$$

7. We can now complete the estimation of the error in the new state, $P_k(+)$, using the Kalman gain matrix, which tells us how much each piece will be trusted.

$$P_k(+) = \left[I - \bar{K}_k H_k\right] P_k(-) \tag{8}$$

8. Finally, we can calculate the a posteriori state estimate using the Kalman gain, the a priori state estimate and the new measurement.

$$\hat{x}_k(+) = \hat{x}_k(-) + \bar{K}_k \left[z_k - H_k \hat{x}_k(-)\right] \tag{9}$$

Kalman Filter Summary

Kalman filter variables:

- Φ_k = process matrix at step k defining how the system changes
- Q_k = process noise covariance matrix at step k
- H_k = measurement matrix relating the state variable and measurement
- R_k = measurement noise covariance matrix at step k
- $\hat{x}_k(+)$ = system state estimate at step k
- P_k = system covariance matrix at step k, estimate of uncertainty in $\hat{x}_k(+)$
- K_k = Kalman gain estimate at step k, how much to trust the measurement

Kalman filter equations

$$\hat{x}_k(-) = \Phi_{k-1}\hat{x}_{k-1}(+) \tag{10}$$

$$P_k(-) = \Phi_{k-1} P_{k-1}(+) \Phi_{k-1}^T + Q_{k-1} \tag{11}$$

$$\bar{K}_k = P_k(-1)H_k^T \left[H_k P_k(-)H_k^T + R_k\right]^{-1} \tag{12}$$

$$P_k(+) = \left[I - \bar{K}_k H_k\right] P_k(-) \tag{13}$$

$$\hat{x}_k(+) = \hat{x}_k(-) + \bar{K}_k \left[z_k - H_k \hat{x}_k(-)\right] \tag{14}$$

Kalman Filter Example: Measuring Temperature (1D problem)

Temperature is a slowly varying signal we can model as a constant value with a small system variance.

Setup

- State is a single value: $[x]$ (temperature in C)
- Estimate a process variance Q (difficult to do, so probably just a small value)
- Estimate a measurement variance R by calculating the standard deviation for a large number of measurements of a constant temperature situation (or even just a constant voltage source). Want to separate the actual process noise and the measurement noise.
- Define a state estimation matrix

  Temperature is modeled as a constant value, so $\Phi = [1]$
- Relationship between measurements and the temperature H

  If we have C comes directly from the measurement process, then $H = [1]$

  If the measurement undergoes a transformation, put that in H
- Let the initial state be $[0]$
- Let the initial error be $[P]$

For each new measurement

- The a priori state is the same as the last a posteriori state (Φ is the identity)
- The a priori error is the same as the a posteriori error plus Q
- The Kalman gain is

$$K_k = \frac{P_k(-)}{P_k(i) + R} \tag{15}$$

- The new a posteriori state is

$$\hat{x}_k(+) = \hat{x}_k(-) + K_k \left(z_k - Hx_k(-)\right) \tag{16}$$

- The new a posteriori error is

$$P_k(+) = \left(1 - K_k H\right) P_k(-) \tag{17}$$

After each iteration, the temperature is updated based on the new measurement. Note that the value of P will converge to a constant, as will K, so long as R and Q are constant. Both P and K can be precomputed use a differential equation, but it can be simpler to run the filter for a while and then preset P and K based on the stable values.

Kalman Filter Summary

Kalman filter variables:

- Φ_k = process matrix at step k defining how the system changes
- Q_k = process noise covariance matrix at step k
- H_k = measurement matrix relating the state variable and measurement
- R_k = measurement noise covariance matrix at step k
- $\hat{x}_k(+)$ = system state estimate at step k
- P_k = system covariance matrix at step k, estimate of uncertainty in $\hat{x}_k(+)$
- K_k = Kalman gain estimate at step k, how much to trust the measurement

Kalman filter equations

$$\hat{x}_k(-) = \Phi_{k-1}\hat{x}_{k-1}(+) \tag{1}$$

$$P_k(-) = \Phi_{k-1}P_{k-1}(+)\Phi_{k-1}^T + Q_{k-1} \tag{2}$$

$$\bar{K}_k = P_k(-1)H_k^T \left[H_k P_k(-) H_k^T + R_k\right]^{-1} \tag{3}$$

$$P_k(+) = \left[I - \bar{K}_k H_k\right] P_k(-) \tag{4}$$

$$\hat{x}_k(+) = \hat{x}_k(-) + \bar{K}_k \left[z_k - H_k \hat{x}_k(-)\right] \tag{5}$$

Kalman Filter Example: Measuring Temperature (1D problem)

Temperature is a slowly varying signal we can model as a constant value with a small system variance.

Setup

- State is a single value: $[x]$ (temperature in C)
- Estimate a process variance Q (difficult to do, so probably just a small value)
- Estimate a measurement variance R by calculating the standard deviation for a large number of measurements of a constant temperature situation (or even just a constant voltage source). Want to separate the actual process noise and the measurement noise.
- Define a state estimation matrix

  Temperature is modeled as a constant value, so $\Phi = [1]$
- Relationship between measurements and the temperature H

  If we have C comes directly from the measurement process, then $H = [1]$

  If the measurement undergoes a transformation, put that in H
- Let the initial state be $[0]$
- Let the initial error be $[P]$

For each new measurement

- The a priori state is the same as the last a posteriori state (Φ is the identity)
- The a priori error is the same as the a posteriori error plus Q
- The Kalman gain is

$$K_k = \frac{P_k(-)}{P_k(i) + R} \tag{6}$$

- The new a posteriori state is

$$\hat{x}_k(+) = \hat{x}_k(-) + K_k \left(z_k - H x_k(-) \right) \tag{7}$$

- The new a posteriori error is

$$P_k(+) = \left(1 - K_k H\right) P_k(-) \tag{8}$$

After each iteration, the temperature is updated based on the new measurement. Note that the value of P will converge to a constant, as will K, so long as R and Q are constant. Both P and K can be precomputed use a differential equation, but it can be simpler to run the filter for a while and then preset P and K based on the stable values.

Kalman Filter Example: Tracking Temperature with Velocity Estimation (1D dependent)

Setup

- State will be the temperature and its derivative (rate of change). The two variables are no longer independent because the temperature at the next time step depends upon the rate of change.
- Estimate a process covariance matrix Q
  - Let the value process have a small error (0.1)
  - Let the derivative process have a larger error (0.5)
  - Note that accelerations, changes in rate of change, are noise under this model
- Estimate a measurement covariance matrix R
  - The measurements might have an estimated error of 1 degree
  - The derivative measurement is the result of differentiating the measurement stream
  - The derivative measurement will have more error, in general
  - Use standard error propagation to estimate the error of the derivative measurement
- Define the state estimation process Φ
  - The value of the temperature changes according to the derivative
  - Assume we are updating the filter at 10Hz, so $\Delta t = 0.1\text{s}$

    $$\Phi = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$$
- Define the relationship between the measurements and the state vector
  - Only measuring temperature directly, derivative is calculated from the measurement stream
  - There is a linear relationship between the binary value representing the voltage and the actual temperature. There will be a single offset that aligns the scaled voltage with the actual temperature. This should be added to the state estimate and only affects interpretation of the meaning of the values.

    $$\begin{bmatrix} z \\ \dot{z} \end{bmatrix} = H \begin{bmatrix} x \\ \dot{x} \end{bmatrix} = \begin{bmatrix} a & 0 \\ 0 & a \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$$
- Define an initial state $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$
- Define an initial error matrix, let it be the identity.

For each new measurement:

Note that we have to execute matrix inverses now since the state vector values are not independent. So the update equations are no longer independent for each state variable.

The P matrix will also end up being a covariance matrix with non-zero off-diagonal terms since in the open loop update equation it is multiplied by the state update matrix.

Genetic Algorithms and Evolutionary Programming

The traditional design of systems has evolved a set of useful design tools

- Mathematical models
- Lumped parameter systems
- Analog system simulators based on the above [SPICE]
- Digital components and design methods
- Discrete-event simulators
- Programming languages
- Modular design

These design tools allow us to design circuits that behave well over a range of temperatures, noise levels, and other external influences

These design tools also allow us to understand, simulate, and trouble-shoot our designs

However, they also limit our ability to create systems

- We abstract away from the physical world to simplify the task
- We stay within a particular domain–digital logic / analog design / functional programming–to keep the system simple, intuitively understandable, and modelable
- This limits the space of possible designs we consider for a particular system

What if we could let the system explore the space of possible designs on its own?

- Biologically motivated
- Evolution is the exploration of design spaces by a self-modifying system
- We can apply evolutionary techniques to parameter selection.
- We can apply evolutionary techniques to algorithms.
- Can we apply evolutionary techniques to physical systems?

Genetic algorithms [GA]

GAs were originally designed as a massively parallel optimization/search method

Given: a problem for which we would like to find the best possible solution. For example, we might have a complex function for which we want to find a minimum or maximum value.

The basic genetic algorithm requires that a solution to a problem be representable as a bit string. For example, if the solution is a number (e.g., a parameter to a system), then we could use the binary representation of that number or a grey code representation.

Parameters

- Population size N
- Crossover rate C
- Mutation rate μ
- Maximum number of generations K

Algorithm

1. Generate a population of N random bit strings, each a potential solution
2. Evaluate all of the bit strings in the population
   - A "Fitness Function" evaluates each individual
   - A better solution should be assigned a higher fitness
3. Stochastically select N strings from the old population to become the next generation
   - Strings with a higher fitness should be more likely to be selected
   - Use a randomized process with the probability of selection proportional to fitness
4. Stochastically select $M \leq N$ strings to participate in crossover (crossover rate C)
   - Stochastically select a location on the bit string
   - Swap the halves so that two new strings are created from the two parents
5. Stochastically mutate some of the bits on strings in the population (mutation rate μ)
   - Set a mutation rate and test each bit in the population to see if it flips
6. Repeat the fitness evaluation and new generation selection (number of generations K)
7. Terminate when the solution is good enough, the population has largely stabilized, or the maximum number of generations K has been reached.

Genetic Algorithms and Evolutionary Programming

The traditional design of systems has evolved a set of useful design tools

- Mathematical models
- Lumped parameter systems
- Analog system simulators based on the above [SPICE]
- Digital components and design methods
- Discrete-event simulators
- Programming languages
- Modular design

These design tools allow us to design circuits that behave well over a range of temperatures, noise levels, and other external influences

These design tools also allow us to understand, simulate, and trouble-shoot our designs

However, they also limit our ability to create systems

- We abstract away from the physical world to simplify the task
- We stay within a particular domain–digital logic / analog design / functional programming–to keep the system simple, intuitively understandable, and modelable
- This limits the space of possible designs we consider for a particular system

What if we could let the system explore the space of possible designs on its own?

- Biologically motivated
- Evolution is the exploration of design spaces by a self-modifying system
- We can apply evolutionary techniques to parameter selection.
- We can apply evolutionary techniques to algorithms.
- Can we apply evolutionary techniques to physical systems?

©2006 Bruce A. Maxwell

Genetic algorithms [GA]

GAs were originally designed as a massively parallel optimization/search method

Given: a problem for which we would like to find the best possible solution. For example, we might have a complex function for which we want to find a minimum or maximum value.

The basic genetic algorithm requires that a solution to a problem be representable as a bit string. For example, if the solution is a number (e.g., a parameter to a system), then we could use the binary representation of that number or a grey code representation.

Parameters

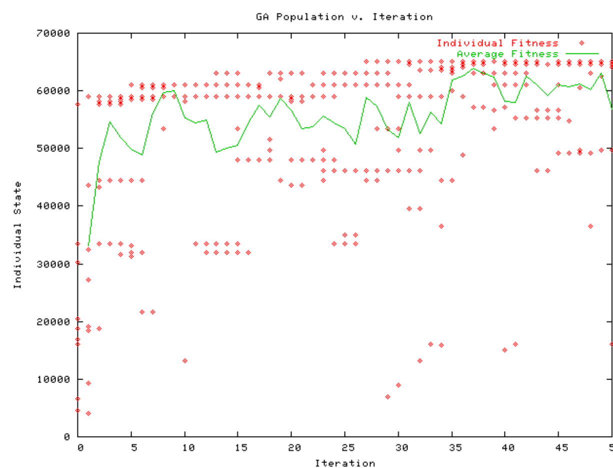
- Population size N
- Crossover rate C
- Mutation rate μ
- Maximum number of generations K

Algorithm

1. Generate a population of N random bit strings, each a potential solution
2. Evaluate all of the bit strings in the population
   - A "Fitness Function" evaluates each individual
   - A better solution should be assigned a higher fitness
3. Stochastically select N strings from the old population to become the next generation
   - Strings with a higher fitness should be more likely to be selected
   - Use a randomized process with the probability of selection proportional to fitness
4. Stochastically select $M \leq N$ strings to participate in crossover (crossover rate C)
   - Stochastically select a location on the bit string
   - Swap the halves so that two new strings are created from the two parents
5. Stochastically mutate some of the bits on strings in the population (mutation rate μ)
   - Set a mutation rate and test each bit in the population to see if it flips
6. Repeat the fitness evaluation and new generation selection (number of generations K)
7. Terminate when the solution is good enough, the population has largely stabilized, or the maximum number of generations K has been reached.

Example

- Standard GA, maximizing the function over the range of 0-255.
- Each individual is 8-bits, interpreted as an 8-bit unsigned binary integer
- Fitness function is $f(x) = x^2$
- Population size is 10
- Crossover probability is 100%, mutation 2%
- 50 iterations



Note: with so few individuals, the speed of convergence is highly dependent upon the initial conditions; in this case, there were several high fitness individuals in the initial population and convergence was fast.

Note also that even as the population converged, there were still a number of low fitness individuals because of crossover/mutation.

Issues with GAs

- Good fitness functions are essential
  - The GA will learn to optimize the fitness function
  - The fitness function should be a continuous function (although it doesnt have to be)
  - If you want robustness to temperature, etc., put that in the fitness function
- The mutation rate is important
  - No mutation limits the search space
  - Too much mutation makes the search random
- The population size is important
  - A larger population size gives you a better sampling of the search space
  - A smaller population makes the process go faster (but does it go to the right place?)

Evolvable Hardware

Hardware circuits form an interesting system

- Subject to physics
- Situated in the real world
- Time-varying components
- Current designs are limited by our tools

Just like any other system, we can evolve hardware circutis

- Set up a physical system
- Develop a bit string representation for it
- Develop a method of evaluating the systems fitness
- Run the GA

Example: Tone Discriminator

Task: discriminate between a 1kHz signal and a 10kHz signal (both square waves)

Physical circuit: Xilinx 6216 programmable logic chip

- Use only one quadrant of the FPGA: 100 blocks in a 10x10 array
- Each block has four inputs and four outputs (N, S, E, W)
- Blocks on the interior have one input and one output
- Blocks on the exterior connect one I/O pin (select a direction)
- Each function block has the following structure
  - Three multiplexors select among the four input signals for F
  - F is a 1x8 memory indexed by the three inputs
- Each output is an inverting MUX that selects among three inputs or F as the output

GA: population = 50, crossover rate = 0.7, mutation rate = 2.7 per generation

Fitness function: maximizes the difference between the average output voltages for a 1kHz signal and a 10 kHz signal

$$f = \frac{1}{5}\left[k_1 \sum_{t \in S_1} i_t + k_2 \sum_{t \in S_2} i_t\right] \quad (1)$$

Note two things about this fitness function:

- It permits small incremental changes in the circuit to lead to improved fitness
- The constants k1 and k2 have to be tuned in order to avoid the identity as a local optimum

Result: after 5000 generations the circuit works perfectly (this was 2-3 weeks of evolution!)

- Always changes correctly on the falling edge of an input waveform, and within 200ns the circuit has made a decision about the frequency of the pulse it just saw.
- Final circuit uses only a small portion of the 10x10 grid (32 blocks)
- Forms three components (A, B, C)
- Parts A and B go inactive during the high part of a pulse
- Part C remains static during the high part of a pulse
- 200ns after a falling edge, the circuit assumes the correct state
- The designers can‘t figure out how it keeps time
- The output is temperature sensitive (but not within the temperatures the cicuit experienced while learning

Evolvable Hardware

Hardware circuits form an interesting system

- Subject to physics
- Situated in the real world
- Time-varying components
- Current designs are limited by our tools

Just like any other system, we can evolve hardware circutis

- Set up a physical system
- Develop a bit string representation for it
- Develop a method of evaluating the systems fitness
- Run the GA

Example: Tone Discriminator

Task: discriminate between a 1kHz signal and a 10kHz signal (both square waves)

Physical circuit: Xilinx 6216 programmable logic chip

- Use only one quadrant of the FPGA: 100 blocks in a 10x10 array
- Each block has four inputs and four outputs (N, S, E, W)
- Blocks on the interior have one input and one output
- Blocks on the exterior connect one I/O pin (select a direction)
- Each function block has the following structure
  - Three multiplexors select among the four input signals for F
  - F is a 1x8 memory indexed by the three inputs
- Each output is an inverting MUX that selects among three inputs or F as the output

GA: population = 50, crossover rate = 0.7, mutation rate = 2.7 per generation

Fitness function: maximizes the difference between the average output voltages for a 1kHz signal and a 10 kHz signal

$$f = \frac{1}{5}\left[k_1 \sum_{t \in S_1} i_t + k_2 \sum_{t \in S_2} i_t\right] \quad (1)$$

Note two things about this fitness function:

- It permits small incremental changes in the circuit to lead to improved fitness
- The constants k1 and k2 have to be tuned in order to avoid the identity as a local optimum

Result: after 5000 generations the circuit works perfectly (this was 2-3 weeks of evolution!)

- Always changes correctly on the falling edge of an input waveform, and within 200ns the circuit has made a decision about the frequency of the pulse it just saw.
- Final circuit uses only a small portion of the 10x10 grid (32 blocks)
- Forms three components (A, B, C)
- Parts A and B go inactive during the high part of a pulse
- Part C remains static during the high part of a pulse
- 200ns after a falling edge, the circuit assumes the correct state
- The designers can't figure out how it keeps time
- The output is temperature sensitive (but not within the temperatures the cicuit experienced while learning

Robot Controller

Task: avoid walls and keeping moving

Physical setup: two sonars, two motors

Electrical system: Dynamic state machine

- 1k by 8bits RAM
  - 10 address input
  - 6 inputs fixed, 6 of 8 outputs used in next address computation)
  - Only 32 bits of information, since the 6 address lines are fixed to specific values
- Optional latches on the RAM addresses (signal can be latched by a clock, or passed through asynchronously) (4 bits)
- Optional latches on two of the RAM outputs going to the motors (2 bits)
- The 2 RAM motor outputs go back to the 10 address inputs plus 2 sonar outputs
- The clock frequency is evolved (16 bit gray code: range 2Hz to several kHz)
- 54 bits to represent a particular DSM

GA: population = 30, crossover rate = 0.7, mutation rate = 1/generation

Fitness function: integrates a value proportional to distance from the wall, and subtracts a value if the robot is stopped (penalizes a stopped robot).

- Fitness of each bit string (configuration) evaluated for 30s
- Towards the end of the training, fitness was evaluated for 90s
- Training took place in a robot virtual reality with the wheels off the ground and simulated sonar inputs
- Noise was added to both in order to simulate reality

Results: after 35 generations the robot shows good performance. The robot used 32 bits of RAM, 3 of the six possible flip-flops, and the clock.

Analysis

Analysis of GA designed systems can be difficult, especially if (as in the tone discriminator) it uses properties of the system that are unexpected or outside lumped parameter models of system components.

- Evolved to use a 9Hz clock (twice the sonar frequency, hmmmm)
- Sonar inputs to the RAM were asynchronous (fairly slow pulse signals)
- Both motor outputs were clocked
- Internal state variable for the left motor was asychronous
- Internal state variable for the right motor was clocked

This is neat: since the motors are clocked, we can think of the state as being sampled by the motors on a 9Hz schedule

The clock allowed the robot to move with a slight waggle, causing the sonars to scan the walls and minimize erroneous sonar readings (hmmm)

Training for robustness I: robot controller with bit errors

Task: same as before, but one of the RAM bits gets incorrectly stuck to a 1 or a 0

Problem: cant feasibly test each individual for 32 different faults (32 x longer to train)

Solution:

- Start by training a population as above for N generations (85)
- After N generations, test the consensus string for all 32 faults and pick the worst. The consensus string is the string with each bit being the most likely for its position given the population.
- Iterate the following
  - Create the next generation, evaluating the fitness function using the current fault
  - Test the consensus individual on all 32 faults and pick the worst as the current fault

By generation 204, the consensus individual displayed complete fault tolerance

Training for robustness II

How do you make a circuit robust to temperature? **The Evolvatron**

- Put one circuit in the freezer
- Put one circuit in a heated environment (next to a light bulb)
- Put one circuit on the table

- Use FPGAs from different batches, and with different cases
- Evaluate the fitness function on all of the different FPGAs in all of the situations

Custom hardware for evolution of buildable circuits: **the Evolvable Motherboard**

- Make a programmable crosspoint architecture that can connect a variety of elements
- The crosspoint architecture also allows you to monitor each signal in the system
- Hook it up to a computer using a bus-based connection (ISA, PCI, etc.)
- Each switch on the crosspoint architecture has some resistance

Example: Inverting amplifier

Task: Evolve an inverting amplifier with a gain of -10
Physical setup:

- EM connecting the bases, emitters, and collectors of transistors
- Switches act as resistors in the circuit

Fitness function: have the output match the amplified input signal

Driving function: 1kHz sine wave, 2mV peak to peak amplitude, offset at 1.4Vdc

GA Characteristics: population = 50, elitism, crossover rate = 1, mutation rate = .01 per bit

- 48 rows/columns
- Each column could have up to 4 connections to a row
- 1056 bits in each string

Results: it worked, using two amplifiers and lots of switches, each of which acted as a resistor.

Example: Evolving in simulation

Task: generate a transistor amplifier

- Have to require the simulator to use standard values for capacitances and resistances
- Have to put in penalties for high collector and emitter currents
- Have to match component parameters to the real world such as saturation current and gain

With these constraints, the GAs trained more slowly, but had a higher probability of working when built out of real components.

Without these constraints, the GAs trained more quickly, but didn‘t create buildable circuits

Intellectual Property Rights

What is Intellectual Property?

- Ideas
- Manifestations of ideas: books, art, catchy phrases
- Processes and Inventions
  - internal combustion engine
  - process for manufacturing integrated circuits
  - person detector
  - combination of sensors in a new and unique configuration
  - digital logic to tie it all together and implement it
- Software (only possible through computers)
  - Algorithm
  - Source code
  - Object code
  - Look & feel

Why do we have intellectual property laws? **In order to promote art and science.**

- Reward people for their innovation
- Ensure that knowlege is distributed

Four legal categories of intellectual property

- Trademark: phrase, name, logo
- Trade Secret: something you dont want to let anyone else know about
- Copyright: manifestations of ideas
- Patent: processes, inventions

Trade Secret

Trade secret laws allow companies to keep secrets.

A trade secret must:

- have novelty
- represent an economic investment to the claimant,
- have involved some effort in development, and
- the company must show that it made some effort to keep the information a secret.

Most well-known example: Coca-Cola
In the technical world: Industrial Light & Magic, Microsoft Windows

Problems:

- Laws are not uniform throughout the U.S. (and definitely not beyond)
- The laws were not written with computer technology in mind
- There is a risk if you go to court in untested territory
- A court could decide that trade secret laws don‘t apply

The *Uniform Trade Secrets Act* has been adopted by about 40 states, and brings a measure of uniformity to trade secret law. The other piece of relevant legislation is the *Economic Espionage Act* of 1996, which makes theft or misappropriation of a trade secret a federal crime. The act applies to both providing secrets to foreign powers and to using secrets for commercial or economic purposes.

Enforcement of trade secrets is generally executed through:

- Non-disclosure agreements [NDA] between an employer and an employee, wherein the employee agrees not to disclose certain information except to specified individuals. This is a contract that is normally required as a condition of employment, and may specify financial penalties in the case of violation.
- Non-competition agreements between an employer and an employee, or between companies, where one party agrees not to use trade secrets–or hire away people who know them–as the basis for competing against the other party. There may be a time limitation on the non-competition clause.

One of the reasons shredders are big business is that they are evidence of a *quality of confidence* regarding the material being shredded. It‘s not unreasonable to piece together a document that has been shredded, but the act of shredding is evidence of trying to keep the secret, which gives the company standing in legal proceedings.

Trade secrets are potentially useful for digital designs

- Integrated circuits can be difficult to reverse engineer
- The lifetime of digital circuits is usually limited, so the secret may not have to be kept for long
- Keeping the design secret will hinder the ability of other companies to create modifications or variations of the design

However, a trade secret does not provide protection from another company designing a circuit with identical overall functionality.

Trade secrets don‘t work so well for software

- A company must reveal the secret to sell the software
- Once the trade secret is known to the relevant public, trade secrecy laws do not apply

Copyright

Copyrights provide the following *exclusive rights* to the copyright holder (taken from Library of Congress copyright web site)

1. To reproduce the work in copies or phonorecords;
2. To prepare derivative works based upon the work;
3. To distribute copies or phonorecords of the work to the public by sale or other transfer of ownership, or by rental, lease, or lending;
4. To perform the work publicly, in the case of literary, musical, dramatic, and choreographic works, pantomimes, and motion pictures and other audiovisual works;
5. To display the work publicly, in the case of literary, musical, dramatic, and choreographic works, pantomimes, and pictorial, graphic, or sculptural works, including the individual images of a motion picture or other audiovisual work; and
6. In the case of sound recordings*, to perform the work publicly by means of a digital audio transmission.

The phrase *exclusive right* means that only the copyright holder is free to exercise the attendant rights, and others are prohibited using the work without the consent of the copyright holder.

Copyright office has been accepting computer program source code since 1964.

In 1980, congress amended the copyright law to explicitly include programs under the category of literary works that are automatically covered without special effort on the part of the author.

Definition: "a set of statements or instructions used directly or indirectly in a computer in order to bring about a certain result."

Copyrights are a weak form of protection, especially for commercially valuable items like software

- Only protects the expression of the idea
- Can write the algorithm in another language
- Can develop the code independently (not copying, but independent development)
- Burden of proof is on the copyright holder to show the intellectual property in question was used as the basis for developing the copy
  - Have to show access to the copyrighted work during development
  - Have to show that the infringing party copied the material

Does copyright extend to the look and feel of a program?

- The structure, sequence, and organization of a program falls under copyright protection
- However, external forces may require similar structure among programs
- *Is the look and feel of a program the idea, and the source and object code the expression?*
  If so, look and feel could not be copyrighted

DMCA

Recent changes in copyright law affect digital transmission and encoding of information. The law is called the Digital Millennium Copyright Act

- You can now be prosecuted for building devices to break encryptions designed to maintain copyrights.
- Certain encryption & digital technologies allowed under the new laws threaten fair use
- Pay per use encryption methods means libraries would have to charge for each use by a borrower
- You would have to pay a price if you wanted to let your friend borrow a document

Some copyright stakeholders have lobbied/are lobbying for copyright to databases that just contain facts (like NBA statistics). Current law is a database is not copyrightable unless it is the result of a creative process.

Fair Use

Copying of copyrighted materials is permitted under certain restricted situations. For example:

- Archival copy of software
- Copying for personal scholarly use (but not for the design of commercial products)
- Copying for classroom use in certain situations (only the first time in many cases)

Whether a use is fair use depends upon four factors

1. The purpose and character of the use, including whether such use is of a commerical nature or is for nonprofit educational purposes
2. The nature of the copyrighted work - highly personal works get more protection
3. The amount and substantiality of the portion used in relation to the copyrighted work as a whole
4. The effect of the use upon the potential market for or value of the copyrighted work

In fair use cases, the four criteria are supposed to be weighted equally. In practice, factors #1 and #4 have received the most attention since it is usually commercial ventures that sue, and they can throw up numbers in support of their case.

Copyright term

- Works created after 1977 receive copyright protection for the life of the author plus 70 years
- Works created for hire receive protection for 95 years from publication or 120 years from creation, whichever is shorter
- Works created before 1978 receive a total duration of protection of 95 years

Protection for copyrighted works prior to 1978 used to be 75 years, but in 1998, Congress extended the life of these copyrighted items by 20 years under pressure from publishers and the recording industry and Disney.

Copyrighted works since 1923 (birth of movie industry) are not in the public domain

In 2002 the Supreme Court heard a case on the 1998 extension that it was unconstitutional because it was no longer a limited copyright as specified in the constitution. It was argued by a Stanford professor, Lawrence Lessig, but lost 7-2.

In Lessig‘s opinion, the five conservative judges who had applied the concept of enumeration of powers in other cases–where they struck down liberal regulations–ignored the enumeration of powers argument that they themselves put forward in the early 80‘s. The enumeration of powers argument is that there are limits to the power of each of the three branches of the federal government, and that these limits are specified implicitly or explicitly by the Constitution. In the case of copyright, the Constitution says:

There is no expectation that Congress will not extend copyrights another 20 years in 2018 unless there is enough public outcry that it becomes a significant factor in elections. Because of this, Lessig believes the era of public domain intellectual property is over.

In response, foundations and groups such as the Free Software Foundation and Lessig‘s Creative Commons are trying to promote alternative methods of intellectual property protection that they believe more appropriately balances interests and promotes the useful arts and sciences.

Intellectual Property Rights

Copyright: Fair Use

Copying of copyrighted materials is permitted under certain restricted situations. For example:

- Archival copy of software
- Copying for personal scholarly use (but not for the design of commercial products)
- Copying for classroom use in certain situations (only the first time in many cases)

Whether a use is fair use depends upon four factors

1. The purpose and character of the use, including whether such use is of a commerical nature or is for nonprofit educational purposes
2. The nature of the copyrighted work - highly personal works get more protection
3. The amount and substantiality of the portion used in relation to the copyrighted work as a whole
4. The effect of the use upon the potential market for or value of the copyrighted work

In fair use cases, the four criteria are supposed to be weighted equally. In practice, factors #1 and #4 have received the most attention since it is usually commercial ventures that sue, and they can throw up numbers in support of their case.

Copyright term

- Works created after 1977 receive copyright protection for the life of the author plus 70 years
- Works created for hire receive protection for 95 years from publication or 120 years from creation, whichever is shorter
- Works created before 1978 receive a total duration of protection of 95 years

Protection for copyrighted works prior to 1978 used to be 75 years, but in 1998, Congress extended the life of these copyrighted items by 20 years under pressure from publishers and the recording industry and Disney.

Copyrighted works since 1923 (birth of movie industry) are not in the public domain

In 2002 the Supreme Court heard a case on the 1998 extension that it was unconstitutional because it was no longer a limited copyright as specified in the constitution. It was argued by a Stanford professor, Lawrence Lessig, but lost 7-2.

In Lessig‘s opinion, the five conservative judges who had applied the concept of enumeration of powers in other cases–where they struck down liberal regulations–ignored the enumeration of powers argument that they themselves put forward in the early 80‘s. The enumeration of powers argument is that there are limits to the power of each of the three branches of the federal government, and that these limits are specified implicitly or explicitly by the Constitution. In the case of copyright, the Constitution says:

> To promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries;

There is no expectation that Congress will not extend copyrights another 20 years in 2018 unless there is enough public outcry that it becomes a significant factor in elections. Because of this, Lessig believes the era of public domain intellectual property is over.

In response, foundations and groups such as the Free Software Foundation and Lessig‘s Creative Commons are trying to promote alternative methods of intellectual property protection that they believe more appropriately balances interests and promotes the useful arts and sciences.

Patents

A patent is the strongest form of protection for intellectual property because it gives the developer a monopoly over the use of that idea for a period of 14 or 20 years.

Main purpose of a patent?

- To encourage the advancement of useful arts and sciences
- Foster invention
- Promote disclosure of inventions
- To ensure the idea is in the public domain

What can be patented?

- Process
- Machine
- Article of manufature
- Composition of matter
- Improvement of any of the above
- Ornamental design of an article of manufacture
- Asexually reproduced plant varieties by design and plant patents
- Gene sequences and chemical compounds that have specific utility
- Software with a specific, limited utility
- Not atomic weapons (Atomic Energy Act of 1954)

What cannot be patented?

- abstract ideas,
- laws of nature,
- physical phenomena,
- literaray, dramatic, musucal and artistic works
- inventions which are not useful or offensive to public morality

The justification is that to give someone the right to these things would inhibit scientific thought and advancement (or be morally incorrect).

A patent claim must:

- fall within the category of permissible subject matter
- have utility
- have novelty
- must be nonobvious
- be adequately described or enabled (for one of ordinary skill in the art to make and use the invention)
- claimed by the inventor in clear and definite terms

What types of patents are there?

- Utility patents may be granted to anyone who invents or discovers any new, useful, and nonobvious process, machine, article of manufacture, or composition of matter, or any new and useful improvement thereof. These are the most common.
- Design patents may be granted to anyone who invents a new, original, and ornamental design for an article of manufacture.
- Plant patents may be granted to anyone who invents or discovers AND asexually reproduces any distinct and new variety of plant.

Provisional v. Non-provisonal patents

The **non-provisional** application establishes the filing date AND initiates the examination process.

The **provisional** application only establishes the filing date and automatically becomes abandoned after one year.

You may file a provisional application when you are not ready to enter your application into the regular examination process. A provisional application establishes a filing date at a lower cost for a first patent application filing in the United States and allows the term ”Patent Pending” to be applied to the invention. Claims are not required in a provisional application. The PTO does not examine a provisional application and such an application cannot become a patent. You must submit the non-provisional application within one year of submitting your provisional application in order to possibly receive the benefit of the provisional application's filing date. You do not have to file a provisional application before filing a non-provisional application.

Since U.S. patent law uses the rule of **first discovery**, establishing a discovery date is essential in order to protect your patent. One way to establish such a discovery date is by filing a provisional patent.

Patent Process

1. Generate a great idea, process, or invention: not particularly easy, often occurs in the shower
2. Figure out if your idea is patentable: most engineering ideas probably are patentable
3. Figure out if it infringes on any other patents and whether the idea is novel: execute a reasonable literature search and a search of the online patent database
4. Start the process with a provisonal application (or not, this step is optional)
5. Develop your non-provisional application, usually with a patent lawyer
   - Title
   - Background of the invention, showing utility
   - Description of the prior art, showing novelty
   - Summary of the invention, giving an overview of inputs, outputs, and the process
   - Brief description of drawings in the application, like a list of figures
   - Detailed description of the invention, which is somewhere between a recipe and a technical paper; the text is often focused around describing the processes outlined in the figures.
   - Claim(s)

     Claims define the invention and are what are legally enforceable. Therefore, they are extremely important. Whether a patent will be granted is determined, in large measure, by the wording of the claims. Claims continue to be important once a patent is granted, because questions of validity and infringement are judged by the courts on the basis of the claims.
   - Abstract
   - Oath or declaration that the ideas are yours and you think they're novel
6. Go through the **challenge** process with a patent examiner
7. You try to make as broad claims as you can
8. The patent examiner tries to minimize your claims (prosecutes the patent)
9. If you patent is granted, then you can claim a patent on your invention
   - You now have the right to license or sell your patent
   - You now have the right to sue someone for infringement
   - You have to pay a small initial fee to apply for and receive the patent. Then you have to pay maintainance fees at 3.5, 7.5, and 11.5 years. The fees go up each time, but they are pretty small ($ 830, $ 1900, and $ 2910 for large organizations, half that for small).
   - In today‘s global commerce, obtaining a patent in the U.S. is not enough in most cases. You also may want to obtain a patent in the EU, Japan, or other nations, each of which requires an additional filing fee.
   - Over $25,000 for a quality patent is not unexpected

Software Patents

Computer software faces its greatest challenge with patent requirement number 1. For the purpose of a patent, software is generally considered a process. In the summary of the invention, for example, a patent application for software processes will use phrasing such as:

> In a fourth exemplary embodiment of the present invention, a computer system is provided. The computer system comprises a CPU and a memory storing an image file. Pursuant to a feature of the present invention, the CPU is arranged and configured to execute a routine to...

The phrasing makes clear that the software is part of the configuration of the computer to execute a process, which is patentable.

70‘s and 80‘s: the argument was software could be duplicated by a mental process, which is inappropriate subject matter, so software patents were rarely granted.

In 1981 the Supreme Court found, in Diamond v. Diehr (1981), that “”an application of a law of nature or mathematical formula to a known structure or process may well be deserving of patent protection.”

In three ensuing cases, the Court of Customs and Patent Appeals set up a test to differentiate patentable from unpatentable subject matter.

Freeman-Walter-Abele test: mathematical algorithms are abstract ideas and unpatentable unless there is a practical application of the algorithm. In previous cases a practical application of an algorithm consisted of data being transformed through mathematical calculations to produce a smooth waveform display on a mointor. According to the test, a practical application required physical elements or steps.

90‘s : new cases focused on how computer algorithms are different from mathematical algorithms, and how abstract a computer algorithm could be and still receive a patent.

Present Law: In State Street Bank & Trust Col v. Signature Financial Group (1998), Inc., the Court of Customs and Patent Appeals held that the transformation of data, representing discrete dollar amounts, by a machine through a series of mathematical calculations into a final share price, constitutes a practical application of a mathematical algorithm, formula, or calculation, because it produces a useful, concrete and tangible result - a final share price.

- The court removed any physical element aspect of patentable software
- The requirement is now that a computer program, method, or process must produce a useful, concrete, and tangible result. If it involves a mathematical algorithm, that’s ok so long as it is the practical application of the algorithm that is patented.

The concern now is that there are too many software patents

- Before you sell a piece of software you have to do a patent search
- The organization of software by the patent office is poor, but better than it was
- There arent enough people at the USPTO to really know whats going on in software
- This makes software development a risky business, and creates barriers to entry