CPU Interfacing with Peripherals

Tajana Simunic Rosing
Department of Computer Science and Engineering
University of California, San Diego.
Bus

- Means for transferring bits
  - wired or wireless
- Bus
  - Connectivity scheme (serial, etc.)
  - Protocol
    - Ports
    - Single function (data bus) or complex protocol (address, data, control)
    - Timing diagrams
    - Arbitration scheme: selecting who gets to transmit when
    - Error detection/correction
Error Detection & Correction

• Error detection:
  – Ability of receiver to detect errors during transmission

• Error correction:
  – Ability of receiver & transmitter to jointly correct the problem

• Parity:
  – Extra bit sent with word used for error detection
  – Even/odd parity: data word plus parity bit transmitted
  – Detects single bit errors, but not all burst bit errors

• Checksum: a count of the number of bits sent in transmission that can be used to check if all bits arrived
Basic Protocol Concepts

- **Actor:** master initiates, slave responds
- **Direction:** sender, receiver
- **Addresses:** special kind of data
  - Specifies a location in memory, a peripheral, or a register within a peripheral
- **Time multiplexing**
  - Share a single set of wires for multiple pieces of data
  - Saves wires at expense of time
I/O Control Methods

Strobe protocol

1. Master asserts *req* to receive data
2. Servant puts data on bus **within time** $t_{access}$
3. Master receives data and deasserts *req*
4. Servant ready for next request

Handshake protocol

1. Master asserts *req* to receive data
2. Servant puts data on bus **and asserts** *ack*
3. Master receives data and deasserts *req*
4. Servant ready for next request
CPU Interface Methods

• Port-based I/O (parallel I/O)
  – Processor has one or more N-bit ports
  – Processor’s software reads and writes a port just like a register

• Bus-based I/O
  – Processor has address, data and control ports that form a single bus
  – Communication protocol is built into the processor
  – A single instruction carries out the read or write protocol on the bus
Types of IO

• Two ways to talk to peripherals
  – Memory-mapped I/O
    • Peripheral registers occupy addresses in the same address space as memory
    • To access I/O use load/store instructions
  – Standard I/O (I/O-mapped I/O)
    • Additional pin (M/IO) on bus indicates whether a memory or peripheral access
    • Special purpose instructions to access
      – e.g. Intel x86 provides in & out instructions

• Most CPUs use memory-mapped I/O
ARM Memory-Mapped I/O

- Define location for device:
  DEV1 EQU 0x1000

- Read/write code:
  LDR r1,#DEV1 ; set up device address
  LDR r0,[r1] ; read DEV1
  LDR r0,#8 ; set up value to write
  STR r0,[r1] ; write value to device
Raspberry Pi-GPIO interface

• GPIO (General-Purpose Input/Output)
  – Memory mapped peripheral
  – Controllable by the user at run time
  – Configured to be input or output

• GPIO in Raspberry Pi 2
  – Physical interface to external sensors
  – 26 GPIO pins and 14 power or ground pins
  – Configured as an interrupt source to the ARM when in the input mode
    • Level-sensitive (high/low)
    • Rising/falling edge configurable
I/O Interface on RPi2

- 40 rail and digital GPIO pins
  - Physical pins are labeled on RPi2
  - Software library, wiringPi, has different labels for GPIO pins

- Simple I/O:
  - GPIO takes a binary sample at the pin
  - Used in our project

- Complex I/O:
  - For sensors with specialized protocols
  - Useful for more complex data
    - E.g. IR camera sensor

wiringPi library: https://projects.drogon.net/raspberry-pi/wiringpi/i2c-library/
Serial Communication on RPI2

- Communication via predefined settings:
  - Speed (baud rate): 115200 bps
  - Bits: 8, 1 stop bit
  - No parity or flow control
  - 3.3V rails

- Communicate via GPIO ports:
  - Dedicated pins for Tx & Rx
RPi2 I²C Bus Communication

• Inter-IC is a 2 wire serial protocol
  – Used by CPU to communicate with peripherals
    • Low cost applications
  – Multi-master, multi-slave bus
  – Uses bidirectional clock (SCL) and data (SDA) lines to communicate
    • To use I2C on RPi2, connect to GPIO pins and leverage wiringPI library

• I²C transmission protocol:
  – 7-bit slave address running at max 100 Kbps
  – Start bit
  – Read/write bit
  – ACK from slave
  – Repeat:
    • Byte of information followed by ACK bit from the other party
  – Stop bit from master ends communication
Polling vs Interrupts

- **Polling**
  - Peripheral intermittently receives data which must be serviced by the processor
  - CPU *polls* to check for data
    - It is very inefficient
    - Hard to do simultaneous I/O

- **Interrupt-driven I/O**
  - Extra pin: if \( \text{Int is 1} \), CPU jumps to ISR
  - “polling” of the interrupt pin is built-into the hardware, so no extra time taken!
Priorities and vectors

- **Priorities** - what interrupt gets CPU first.
  - **Masking**: interrupt with priority lower than current priority is not recognized until pending interrupt is complete.
  - **Non-maskable interrupt (NMI)**: highest-priority, never masked; often used for power-down.

- **Vectors**
  - what code is called for each type of interrupt.

- Most CPUs provide both
Assume priority selection is handled before this point.

- CPU acknowledges request.
- Device sends vector.
- CPU calls handler.
- Software processes request.
- CPU restores state to foreground program.
Sources of interrupt overhead

- Handler execution time.
- Interrupt mechanism overhead.
- Register save/restore.
- Pipeline-related penalties.
- Cache-related penalties.
Direct Memory Access

- Buffering
  - Temporarily storing data in memory before processing
- Microprocessor could handle this with ISR
  - Storing and restoring microprocessor state inefficient
  - Regular program must wait
- DMA controller more efficient
  - Separate single-purpose processor
  - Microprocessor relinquishes control of system bus to DMA controller
  - Microprocessor can meanwhile execute its regular program
Arbitration

- Determine which of multiple peripherals gets service first from single resource (e.g., microprocessor, DMA)
  - Daisy chain
  - Network oriented

- **Priority arbiter**
  - Single-purpose processor
  - Peripherals communicate with arbiter
1(a): μP is executing its main program. It has already configured the DMA ctrl registers.

1(b): P1 receives input data in a register with address 0x8000.
Peripheral to memory with DMA

2: P1 asserts `req` to request servicing by DMA ctrl.

3: DMA ctrl asserts `Dreq` to request control of system bus
Peripheral to memory with DMA

4: After executing instruction 100, $\mu$P sees $Dreq$ asserted, releases the system bus, asserts $Dack$, and resumes execution, $\mu$P stalls only if it needs the system bus to continue executing.
Peripheral to memory with DMA

5: DMA ctrl (a) asserts ack, (b) reads data from 0x8000, and (c) writes that data to 0x0001.

(Meanwhile, processor still executing if not stalled!)
Peripheral to memory with DMA

6: DMA de-asserts \( Dreq \) and \( ack \) completing the handshake with P1.

- Program memory
  - No ISR needed!
  - ... Main program ...
  - 100: instruction
  - 101: instruction

- \( \mu P \)
  - \( PC \)
  - 100

- DMA ctrl
  - 0x0001
  - 0x8000

- Data memory
  - 0x0000
  - 0x0001

- System bus
  - 0

- P1
  - 0x8000
Raspberry Pi2 DMA

- Supports 16 DMA channels
  - The first channel is reserved by GPU.
- Mapped from 0x3f007000 in physical memory
  - First chan.: 0x3f007000, second chan.: 0x3f007100, ....
- Accessed and implemented via memory mapped IO (mmap)
- Controls use a control block (CB) structure
  1. Allocate a CB, fill with source & destination addresses
  2. Fill the header of the channel to activate the DMA controller
  3. Copy is operated by the DMA controller (while CPU is doing other tasks). Once done, issues an interrupt to CPU

<table>
<thead>
<tr>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0 CS</td>
<td>Control and status (e.g., reset, activate)</td>
</tr>
<tr>
<td>0x4 CONBLK_AD</td>
<td>Address of the control block</td>
</tr>
</tbody>
</table>
ADC & DAC

Tajana Simunic Rosing
Department of Computer Science and Engineering
University of California, San Diego.
Hardware platform architecture
Sensors and Actuators

- **Sensors**
  - Capture physical stimulus
  - Convert it to electrical signal

- **Actuators**
  - Create physical stimulus
  - Given electrical signals
  - Examples:
    - Pneumatic systems, IR, thermal, motors, MEMS

- **Need analog to digital and digital to analog converters**
  - **Analog**: generate a voltage or current difference that must be measured and processed
    - e.g. ambient light
  - **Digital**: sensors directly generate a digital value
    - e.g. GPS
Embedded System Hardware
Interfacing Sensors and Actuators
“Real World” Sampled Data Systems Consist Of ADCs and DACs

ADC SAMPLED AND QUANTIZED WAVEFORM

DAC RECONSTRUCTED WAVEFORM
Signal Sampling

- **Sampling** converts a *continuous time* signal into a *discrete time* signal
  - It replicates spectrum of continuous-time signal at multiples of the sampling frequency
- **Categories:**
  - Impulse (ideal) sampling (infinitely narrow pulse)
  - Natural sampling (finite width pulse)
  - Sample and hold circuit
Sample and Hold

$V_e$ is analog input signal
$V_e$ is digital sampled signal
**Sampling Methods Examples**

Figure 2.14  Amplitude and time coordinates of source data. (a) Original analog waveform. (b) Natural-sampled data. (c) Quantized samples. (d) Sample and hold.
Impulse Sampling

\[ x(t) = \sum_{n = -\infty}^{\infty} \delta(t - nT_s) \]

\[ x_s(t) = x(t) x_\delta(t) \]

\[ X_\delta(f) = \frac{1}{T_s} \sum_{n = -\infty}^{\infty} \delta(f - nf_s) \]

\[ |X_s(f)| \]
The Aliasing Effect

$\text{If } f_s > 2f_m$,

$\text{Filter characteristic to recover waveform from sampled data}$

$|X_s(f)|$

(a)

$\text{If } f_s < 2f_m$,

Aliasing happens

$|X_s(f)|$

(b)
Ideal Sampling and Aliasing

- Sampled signal is discrete in time domain with spacing $T_s$
- Spectrum will repeat for every $f_s$ Hz
- Aliasing (spectral overlapping) if $f_s$ is too small ($f_s < 2f_m$)
- Sample at least at sampling rate $f_s = 2f_m$
- Generally oversampling is done $\rightarrow f_s > 2f_m$
Nyquist theorem

- Analog input can be precisely reconstructed from its output, provided that sampling proceeds at $\geq$ double of the highest frequency found in the input [Nyquist 1928]

Does not capture the effects of quantization: Quantization noise prevents precise reconstruction.
Quantization

- Quantization is done to make the signal amplitude discrete
Quantization Noise

- quantization noise = approx - real signal

* [http://www.beis.de/Elektronik/DeltaSigma/DeltaSigma.html]
Linear Quantization

\[(L-1) q = 2V_p = V_{pp}\]

For large \(L\)

\[Lq \approx V_{pp}\]
Linear Pulse Code Modulation

\[ V_{\text{max}} = 7.5V \]

Pulse Code Modulation is a method used to digitally represent sampled analog signals

**Linear PCM:**
- Quantization levels are uniform (linear)
- Defined by a sampling rate & bit depth \( L \) (total # of values that can be represented)
Non-Uniform Quantization

• In speech signals, very low speech volumes predominates
  – Voltage exceeds the RMS value only 15% of the time

• These low level signals are under represented with uniform quantization
  – Same noise power but low signal power

• The answer is non uniform quantization
Figure 2.18  Uniform and nonuniform quantization of signals.
Non-uniform Quantization

Compress the signal first
Then perform linear quantization
→ Result in nonlinear quantization

Figure 2.19  (a) Nonuniform quantizer characteristic. (b) Compression characteristic. (c) Uniform quantizer characteristic.
μ-law and A-law

Widely used compression algorithms

Figure 2.20  Compression characteristics. (a) μ-law characteristic.  
(b) A-law characteristic.
PCM: $\mu$ & A-law

- Non-linear PCM encodings are used where quantization levels vary as a function of amplitude
- A-law algorithm has less proportional distortion for small signals
- $\mu$-law algorithm has a slightly larger dynamic range
Embedded System Hardware
Interfacing Sensors and Actuators

A/D converter
sample-and-hold

sensors

environment

information processing

display

D/A converter

actuators
Digital to Analog Converters

• Ideal sampling would allow us to reconstruct the signal perfectly with a sequence of impulses
  – But there is no ideal sampling, so...

• We use zero-order hold circuit to create an analog output
  – Holding each sample value for one sample interval causes multiple harmonics above the Nyquist frequency
  – A low pass reconstruction filter is needed to recreate the signal

Binary-weighted DAC

![Diagram of Digital to Analog Converter](image)
DAC: key parameters

• Resolution
  – # of possible output levels (n bit DAC gives \(2^n\) levels)

• Maximum sampling rate
  – Defined by the Nyquist theorem

• Monotonicity
  – Ability of a DAC's analog output to move only in the direction that the digital input moves; e.g. if the input increases, the output doesn't dip too early

• Total harmonic distortion and noise
  – A measurement of the distortion and noise introduced to the signal by the DAC.
  – A percentage of the total power of unwanted harmonic distortion and noise that accompany the desired signal.

• Dynamic range
  – A measurement of the difference between the largest and smallest signals the DAC can reproduce expressed in decibels.
Sensors & Actuators

System Energy Efficiency Lab

seelab.ucsd.edu
Sensor Classification & Common Issues

- **Local vs. global**
  - part of a device vs. external to the device, but sending data to it

- **Passive vs. active**
  - monitor environment without disturbing it (e.g. thermal) vs. disturb environment (e.g. sonar)

- **Internal vs. external**
  - monitor devices internal state vs. monitor environment

- **Common sensor issues:**
  - Sensitivity to other parameters (e.g. temperature), processing overhead, drift, noise, power, accuracy, calibration, latency, jitter
Smartphone Built-in Sensors

- Proximity sensor
- GPS, A-GPS
- Ambient light
- Gyroscope
- Environment – air temperature, pressure, humidity
- Capacitive/resistive touch
- Camera
- RFID/NFC
Sensing Position

- **Sonar**
  - Emits a short acoustic signal at ultrasonic frequency (50-250kHz) and measures time for echo to return
  - 15° per sensor -> need multiple sensors to get 360°
- **Laser**
  - Provide a perfect local 2/3D map
  - Bulky, heavy, and expensive
- **Infrared**
  - Pulsed infrared LED at 40kHz with a detection array
  - Angle of detection changes with distance to the object -> provides a measure of distance
  - Non-linear -> requires post processing & calibration
  - Not accurate at distances below 6cm -> use IR proximity switch which returns a logic value zero if there is free space short distance in front of it
- **Compass**
- **GPS**: accurate but only outdoors, high energy cost
Accelerometer

- Measures acceleration along one axis
  - Motion triggers displacement of mass, creates a change in capacitance
  - Sensor measures the change in capacitance
  - Microelectromechanical device
  - Multiple sensors provide different axes
- Issue: jitter
Gyroscope

- Gyroscope
  - Measures a change in rotation along one axis (not the absolute position)
  - Issue: drift
  - E.g. iPhone gyro is MEMs based
    - Gyroscope has a plate, called the "proof mass," that vibrates when a drive signal is applied to set of drive capacitor plates. When a user rotates the phone, the proof mass is displaced due to Coriolis forces. CPU senses the proof mass' displacement through capacitor plates located underneath the proof mass, as well as finger capacitors at the edges of the package.
Inclinometer

- Inclinometer (tilt sensor)
  - Measures absolute orientation angle to one axis
  - Better for orientation measurements than gyroscopes
  - Issue: time lag and oscillations

When the sensor is tilted, the capacitance between fixed and flexible (proof mass) electrodes changes and provides a measure of the angle.
More Sensor Examples

- **Electrochemical**
  - Active electrode exposed to gas (or liquid), reference electrode insulated
  - Change in current between the two is proportional to fractional volume → translates to concentration

- **Biosensor**
  - Receptor (ligand) binds to target molecules, causing electrical or chemical change

- **MOS Sensor**
  - Clean air causes adsorption of donor electrons, preventing current flow
  - Presence of other gases reduces surface density of oxygen, increasing current flow
Bigger Sensor: Motor’s Shaft Encoder

- Encoders can use magnetic or optical sensors
  - Magnetic sensors use a Hall effect sensor and a number of magnets mounted in a circle (determines resolution)
  - Optical sensors use an LED to generate light and a photo diode that detects reflections during a white segment but not during the black one -> enables estimate of speed
- It can detect speed of rotation and direction, with additional sensors can also detect the absolute position of the motor shaft

![Diagram of optical encoder](image1)

Optical encoder: detects speed & rotation direction

![Diagram of gray encoder](image2)

Gray encoder: detects absolute position
Simple & Complex Sensing with RPi2

- Simple sensing: read in a single bit on general purpose I/O pin:
  - Sensors provide pull-up (digital HIGH) or pull-down (digital LOW) values, typically over a resistor
  - e.g. button sensor in series with a resistor

- Complex sensing: rely on protocols to send/receive data
  - Sensors accompanied by microcontroller to read/write data as appropriate.
  - Example: RPi2 IR camera
Analog sensing

- Sensors detect over a continuous range, translating into voltage or current changes
- Analog-to-Digital converters (ADC) interpret changes into digital signals – number of bits in ADC reflects resolution:
  - e.g. 16-bit ADC = $2^{16}$ unsigned resolution, $2^{15}$ signed resolution
- Input quantization introduces error
- Sampling rate limits data bandwidth
Analog Sensing on RPi2

- External ADC (ex: MCP3008), digital inputs passed into RPI
- Communication via protocols such as SPI
RPi2 Digital Sensors

- **Key Switch Module (KY-004)**
  - Provide $V_{DD}$ (digital HIGH) and ground
    - Toggle closes the circuit across 10K resistor
    - Produces a digital HIGH when switch is toggled, LOW otherwise

![Diagram of Key Switch Module](image)
RPi2 Digital Sensors

- 2-Color LED Module (KY-011), 3-Color LED Module (KY-016)
- 2 or 3 co-situated light-emitting diodes (LEDs)
  - Similar to RGB display
- Digital high to each of the positive signals connects 3.3V to ground, illuminating the LED

![LED Module Diagram]
**RPi2 Digital Sensors**

- Digital Temperature Module (KY-028)

- Temperature-sensitive diode
- Threshold for sensitivity controlled by potentiometer (controllable resistor)
- Pass 3.3V through to output when threshold is met, GND otherwise
RPi2 Digital Sensors

- Hunt/Tracking Module (KY-033)
  - Infrared (IR) transmitter periodically sends fixed-frequency transmission
  - IR receiver (IR-sensitive diode) tuned to that frequency looks for reflected signal
    - 2-40cm, depending on material
    - Passes through $V_{DD}$ when receives signal
RPi2 Digital Sensors

- Vibration Switch Module (KY-002)
  - Sensitive trigger (Gaoxin switch)
    - Fixed (post) terminal and movable (spring) terminal
    - Sufficient force shorts terminals together
RPi2 Digital Sensors

- Active Buzzer Module (KY-012)

- Piezoelectric effect – materials generate electric potential (voltage) when their shape changes
  - The reverse is also true – apply potential $\rightarrow$ shape change
  - Changing shape frequently $\rightarrow$ noise
TCN75A: Analog Sensor with a built-in ADC

- TCN75A Temperature Sensor Overview
  - Reports from -40°C to 120°C
  - Analog sensor changes current based on perceived temperature
  - Programmable 9- to 12-bit ADC reads the value and generates serial output

- Simple explanation, but more complicated in practice
  - How does the bit width of the ADC get set?
  - How is the output read?
  - How is the system configured and used with an RPi?
TCN75A: Configuration

- Only three I/O ports!
  - 6 if used with multiple other components on the same bus (for addressing)
  - Wired to ground when used as the only device
  - Software-defined communication
  - I²C bus protocol
TCN75A: Communication

- Transmissions written on the rising clock edge
- Need to maintain data before the rising edge (setup time) and after falling edge (hold time)
- When writing to device, determine which register to modify
  - 0x00: data register (read-only)
  - 0x01: configuration register
    - Set ADC bit width
TCN75A: Usage

- Maintaining timing, frequency, reading data manually is hard
- Use the I²C library to manage bookkeeping functionality
  - Ensures transmissions occur on the clock boundaries
  - Write transmissions
  - Read transmissions
- In setup():
  - Set the resolution in a single transaction
  - Reset to the 0x00 register to be ready to read data
- In loop():
  - Read the lower 8 bits (bus limit)
  - Read the upper 4 bits
  - Convert reading to temperature

```cpp
void setup(){
  Serial.begin(9600);
  Wire.begin();
  // Do some setup for the sensor
  // Set the resolution of the measurement
  Wire.beginTransmission(address);
  // point to Configuration Register
  Wire.write(0x01);
  // set the resolution
  Wire.write(0x60);
  // ends the command
  Wire.endTransmission();
  // points to the Temperature Register
  Wire.beginTransmission(address);
  Wire.write(0x00);
  Wire.endTransmission();
}

void loop(){
  // Receives data from the Temperature Register
  Wire.requestFrom(address, byte(2));
  tempreg = Wire.read();
  tempreg= tempreg << 8;
  tempreg |= Wire.read();
  tempreg = tempreg >> 4;
  // Calculate the temperature
  temperature = (float) tempreg / 16;
}
Actuators

- Cause changes to the physical world
  - Can be as passive as changing screen elements, or keeping an aerial system aloft

- Ensure stability/safety/reliability of the embedded system itself
  - Typically feedback loop (e.g. flight stability via PID controller)
  - MoC transitions/states can also trigger actuation
Analog Actuation: Audio Playback

- Digital signals representing MIDI, WAV notes
  - 8-bit DAC converts input into up to 256 distinct voltages

- Varying voltages across simple speaker produces different tones

- Simple actuation:
  - Accomplished without communications protocols
  - More complex actuators (LCD displays, more sophisticated IoT devices) require more complex control
DC Motors

- Revolve freely – need feedback control via shaft encoder
  - stepper motors do not move freely!
  - Efficiency is related to angular shaft velocity $\omega$, applied torque $\tau_a$, input voltage and current

$$\eta = \frac{P_o}{P_i} = \frac{\tau_a \omega}{V_a i}$$
DC-Motor Control: H-Bridge & PWM

- H-Bridge controls the motor’s direction of spinning, and starts/stops it
  - X! = Y: set rotation direction
  - X = Y: stop the motor
- Pulse-Width Modulation
  - Motors are slow: the equivalent Veff controls its speed
  - Pulse-width-ratio is the duty cycle: \( t_{on}/t_{period} \)
  - Non-linear relationship
  - Calibration needed
Servo & Stepper Motors

- Accurate-control motors
  - Driven by PWM control signals
  - **Steppers**: pulses drive armature to next pole (open loop) – e.g. linear actuator
  - **Servos**: Frequency of PWM defines an armature position (closed-loop) – e.g. aeronautical control surfaces
Example Research Application: CitiSense

- Air quality monitoring project in UCSD CSE

- Electrochemical **sensors**, **microcontroller** for data collection and transmission to an **Android** app

- Actuation: air quality is immediately reported, as well as retransmitted to a backend for larger-scale analysis
Sample Commercial Applications

- **Healthcare:** Patient monitoring and reporting
  - Wearable, implantable sensors
  - Artificial organ actuation
  - Fault-tolerant, mission-critical embedded platform

- **Vehicle control:** Airplanes, automobiles, autonomous vehicles
  - All kinds of sensors to provide accurate, redundant view of the world
  - Actuation is maintaining control of the vehicle
  - Very tight timing constraints and requirements enforced by the platforms
Basics of Control

Tajana Simunic Rosing
Department of Computer Science and Engineering
University of California, San Diego.
Closing the Loop: Control

A/D converter
sample-and-hold

sensors

information processing

D/A converter

display

actors

environment
Control System

• **Objective**: output tracks a reference even in the presence of measurement noise, model error and disturbances

• **Metrics**
  – Stability - Output remains bounded
  – Performance - How well an output tracks the reference
  – Disturbance rejection – Tolerate outside error sources
  – Robustness - Ability to tolerate modeling error of the plant

• **Software** gives commands to meet a setpoint, the system responds; E.g: Thermostat, Aircraft altitude control

![Diagram showing overshoot and ripple](image)
Performance

• Rise time
  – Time it takes form 10% to 90%

• Peak time

• Overshoot
  – Percentage by which Peak exceed final value

• Settling time
  – Time it takes to reach 1% of final value
Closed Loop Control Systems

- **Plant** - Physical system to be controlled
  - Car, plane, disk, heater,...
- **Actuator** - device to control the plant
  - Throttle, wing flap, disk motor,...
- **Controller** - Designed product to control the plant
- **Sensor** measures output - aspect of the physical system we are interested in
  - Speed, disk location, temperature
- **Reference or setpoint** - value we want to see at the output
  - Desired speed, desired location, desired temperature
- **Disturbance** - uncontrollable input to the plant imposed by environment
  - Wind, bumping the disk drive, door opening
- **Error detector** – difference between the observed value and setpoint
  - Use a feedback control system to minimize tracking error
Controller Design: on-off control

• If the system is below a set point, then control is turned-on, else it is turned-off
• The difference between on and off is deliberately small, known as the hysteresis $H$, to prevent noise from switching control rapidly and unnecessarily when near the set-point.
• Used by almost all domestic thermostats

Figure source: http://newton.ex.ac.uk/teaching/CDHW/Feedback/ControlTypes.html#OnOffCtl
Controller Design: Proportional Ctrl.

• Good alternative to on-off control
• Signal becomes proportional to the error
  – $P (\text{setpoint} - \text{output})$
  – Example, car speed for cruise control
• Need to find out value of constant $P$
  – Tuning the controller is **hard**
  – What happens if $P$ is too low/high?
• Typically a proportional controller decreases response time so it quickly gets to the setpoint but it increases overshoot
Controller Design: P

- Make a change to the output that is proportional to the current error
  - $P_{out}$: Proportional term of output
  - $K_p$: Proportional gain
  - $e$: Error = Setpoint – Current Value

- P controller often has a permanent offset from setpoint
  - retains error that depends on $K_p$ & the process gain as it needs non-zero error to drive it

- System can become unstable when $K_p$ is too high

$$P_{out} = K_p e(t)$$

Source: wikipedia
P-only Control

• For an open loop overdamped process as $K_p$ is increased the process dynamics goes through the following sequence of behavior
  – overdamped
  – critically damped
  – oscillatory
  – ringing
  – sustained oscillations
  – unstable oscillations
Dynamic Changes as $K_p$ is Increased

![Graphs showing dynamic changes with increasing $K_p$.]
Adding Derivative Control

• To reduce overshoot/ripple, take into account how fast are you approaching the setpoint
  – If very fast, overshoot may be forthcoming: reduce the signal recommended by the proportional controller
  – If very slow, may never get to setpoint: increase the signal

• Proportional-Derivative controllers are slower than proportional, but have less oscillation, smaller overshoot/ripple
Controller Design: Derivative

- Reduces the magnitude of the overshoot produced by the integral component and improves the combined controller-process stability.
- Proportional to the derivative of the error.
- Large $K_d$ decreases the overshoot but amplifies the noise in the signal.

$$D_{out} = K_d \frac{de}{dt}$$

Source: wikipedia
Integral Control

• There may still be error in the PD controller
  – For example, the output is close to the setpoint
    • Proportional is very small and so is the error, discretization of signal will provide no change in the proportional controller
    • Derivative controller will not change signal, unless there is a change in output

• Take the sum of the errors over time, even if they’re small, they’ll eventually add up
Controller Design: Integral

- Proportional to both the magnitude of the error and the duration of the error.
- Large $K_i$ eliminates steady state errors faster but can cause overshoot.

\[
I_{out} = K_i \int_{0}^{t} e(\tau) \, d\tau
\]
Controller Design: PID

• Combine Proportional, Integral, and Derivative control to change Manipulated Variable (MV)
  – Use P to control the amount of disturbance (error)
  – Use D to control the speed of reduction in error
  – Use I to ensure steady state convergence and convergence rate

• Does not guarantee optimality or stability, is not adaptive

\[
MV(t) = P_{out} + I_{out} + D_{out}
\]

\[
u(t) = MV(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de}{dt}
\]
Controller Design: PID

- Effects of *increasing* parameters independently

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Rise time</th>
<th>Overshoot</th>
<th>Settling time</th>
<th>Steady-state error</th>
<th>Stability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_p$</td>
<td>Decrease</td>
<td>Increase</td>
<td>Small change</td>
<td>Decrease</td>
<td>Degrade</td>
</tr>
<tr>
<td>$K_i$</td>
<td>Decrease</td>
<td>Increase</td>
<td>Increase</td>
<td>Eliminate</td>
<td>Degrade</td>
</tr>
<tr>
<td>$K_d$</td>
<td>Minor change</td>
<td>Decrease</td>
<td>Decrease</td>
<td>No effect in theory</td>
<td>Improve if small</td>
</tr>
</tbody>
</table>

Source: wikipedia
PID Controller Tuning

**Model-based**
1. Direct Synthesis
2. Internal Model Control (IMC)
3. Controller tuning relations
4. Frequency response techniques
5. Computer simulation

All these methods relay on off-line model design

**On-Line Tuning**
1. Continuous Cycling
2. Relay Auto-Tuning
3. Step Test Method
   - When in steady state, apply a small test step & set controller reaction settings by process reaction curve

Normally done after initial settings are created with model-based methods
Ziegler and Nichols (1942) introduced the *continuous cycling method* for controller tuning that is based on the following procedure:

- **Step 1.** After the process has reached steady state approximately, eliminate the integral and derivative control actions by setting

  \[ K_d = K_i = \text{zero} \]
• **Step 2.** Set $K_p$ equal to a small value (e.g., 0.5) and place the controller in the automatic mode.

• **Step 3.** Gradually increase $K_p$ in small increments until continuous cycling occurs. The term *continuous cycling* refers to a sustained oscillation with a constant amplitude.
  
  – *Ultimate gain, $K_u$* - The value of $K_p$ that produces continuous cycling for proportional-only control
  
  – *Ultimate period, $T_u$* - The period of the corresponding sustained oscillation
Ziegler and Nichols: Steps 4 & 5

• **Step 4:** Use Ku and Tu to set the gains Kp, Ki & Kd, with table below:

<table>
<thead>
<tr>
<th>Control Type</th>
<th>$K_p$</th>
<th>$K_i$</th>
<th>$K_d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td>$0.5K_u$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$PI$</td>
<td>$0.45K_u$</td>
<td>$1.2K_p/T_u$</td>
<td>-</td>
</tr>
<tr>
<td>$PD$</td>
<td>$0.8K_u$</td>
<td>-</td>
<td>$K_pT_u/8$</td>
</tr>
<tr>
<td>classic $PID$</td>
<td>$0.60K_u$</td>
<td>$2K_p/T_u$</td>
<td>$K_pT_u/8$</td>
</tr>
</tbody>
</table>

• **Step 5.** Fine-tune by introducing a small set-point change and observing the closed-loop response.
Relay Auto-Tuning

- Developed by Åström and Hägglund (1984)
- A simple & easily automated experimental test to get $K_u$ & $T_u$:
  - The feedback controller is temporarily replaced by an on-off controller or relay with amplitude $d$
  - After the control loop is closed, the controlled variable exhibits a sustained oscillation of period $T_u = P$ & $K_u = 4d / \pi a$
Sources

• Real-time DSP Lab, Prof. Brian Evans, UTA
• Ryerson Communications Lab, X. Fernando
• Daniel Mosse & David Willson
• Charles Williams;
  http://newton.ex.ac.uk/teaching/CDHW/Feedback/