LCD Display

CHENGMING ZHANG

2017.3.28

Outline

- Introduction
- Characteristics
- Interfacing

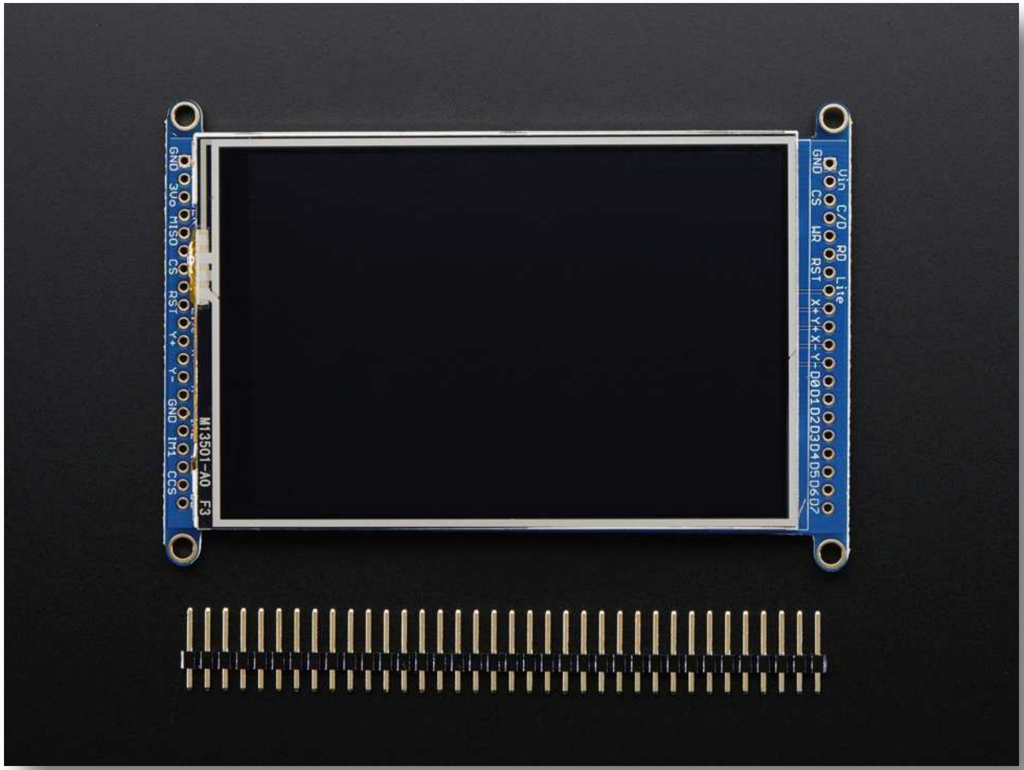
Introduction

- Widely used in daily life(and embedded system as well)
- LCD, LED (with LCD), OLED
- Volatile or static

LCD Characteristics

- Lightweight, compact, portable, cheap
- Use a thin layer of liquid crystal between plate
- Behavior change under different voltage
- Circuit needed to control every part of display

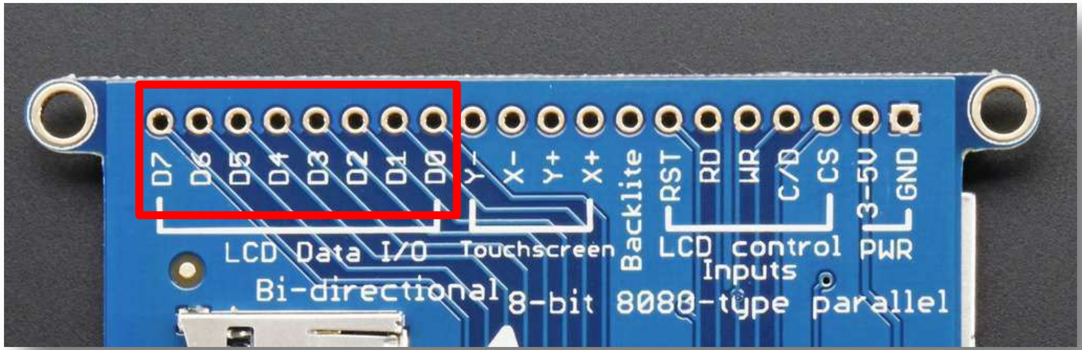
Interfacing

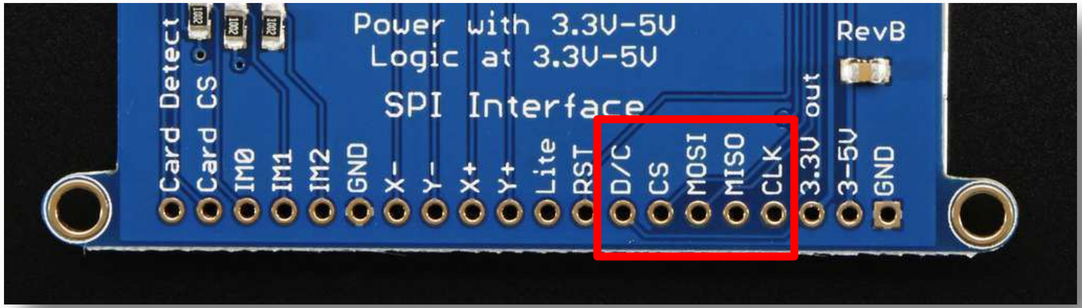


source: https://cdn-shop.adafruit.com/970x728/2050-00.jpg

Interfacing

- 8-pin & SPI mode





Configurations

Name	Description
GND	Ground
3-5v	Power in
MOSI	Master out slave in
MISO	Master in slave out
CS	Select signal
CLK	Clock signal
D/C	Indicating incoming transaction is data or command

Sending Data & Command

- Sending Data
  - D/C high
  - CS high
- Sending Command
  - D/C low
  - CS high

6.2.91 SETCOLOR: set color (EBh)

EBh	SETCOLOR (Set Color)												
	DNC	NRD	NWR	D15~D8	D7	D6	D5	D4	D3	D2	D1	D0	HEX
Command	0	1	↑	-	1	1	1	0	1	0	1	1	EB
1st parameter	1	1	↑	-	Bkx1	Bkx0	Bky1	Bky0	Wx1	Wx0	Wy1	Wy0	-
2nd parameter	1	1	↑	-	Bkx9	Bkx8	Bkx7	Bkx6	Bkx5	Bkx4	Bkx3	Bkx2	-
3rd Parameter	1	1	↑	-	Bky9	Bky8	Bky7	Bky6	Bky5	Bky4	Bky3	Bky2	-
4th Parameter	1	1	↑	-	Wx9	Wx8	Wx7	Wx6	Wx5	Wx4	Wx3	Wx2	-
5th Parameter	1	1	↑	-	Wy9	Wy8	Wy7	Wy6	Wy5	Wy4	Wy3	Wy2	-
6th Parameter	1	1	↑	-	Rx1	Rx0	Ry1	Ry0	Gx1	Gx0	Gy1	Gy0	-
7th Parameter	1	1	↑	-	Rx9	Rx8	Rx7	Rx6	Rx5	Rx4	Rx3	Rx2	-
8th Parameter	1	1	↑	-	Ry9	Ry8	Ry7	Ry6	Ry5	Ry4	Ry3	Ry2	-
9th Parameter	1	1	↑	-	Gx9	Gx8	Gx7	Gx6	Gx5	Gx4	Gx3	Gx2	-
10th Parameter	1	1	↑	-	Gy9	Gy8	Gy7	Gy6	Gy5	Gy4	Gy3	Gy2	-
11th Parameter	1	1	↑	-	Bx1	Bx0	By1	By0	Ax1	Ax0	Ay1	Ay0	-
12th Parameter	1	1	↑	-	Bx9	Bx8	Bx7	Bx6	Bx5	Bx4	Bx3	Bx2	-
13th Parameter	1	1	↑	-	By9	By8	By7	By6	By5	By4	By3	By2	-
14th Parameter	1	1	↑	-	Ax9	Ax8	Ax7	Ax6	Ax5	Ax4	Ax3	Ax2	-
15th Parameter	1	1	↑	-	Ay9	Ay8	Ay7	Ay6	Ay5	Ay4	Ay3	Ay2	-
	Bkx[9:0]:												

- Various command: SETCOLOR SETIMAGE…

Interfacing

- Arduino library available for both 8-bit and SPI mode
- Written in C++, can be ported to c language

Reference

https://en.wikipedia.org/wiki/Flat_panel_display#Plasma_panels

https://www.adafruit.com/product/2050

Question

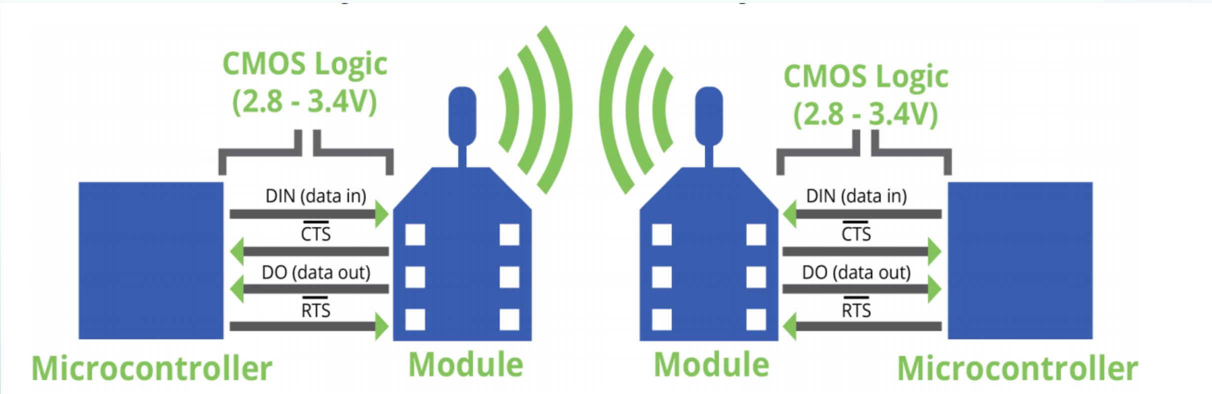
Thank you

RF Module and Sensors

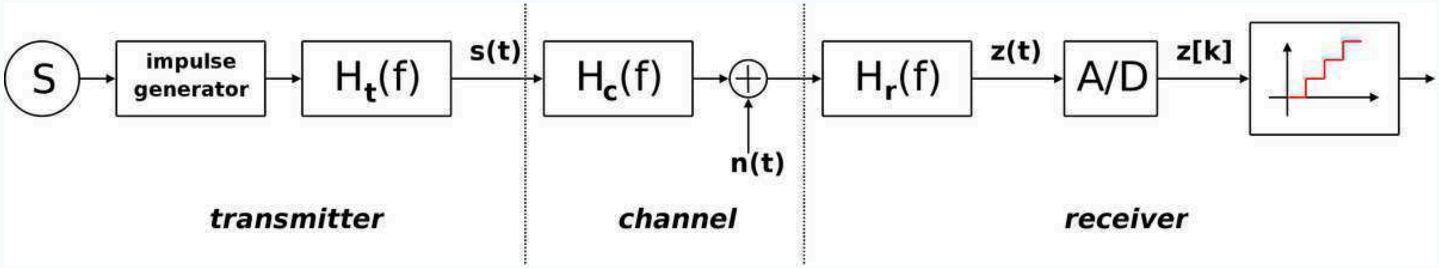
Chunke Tan

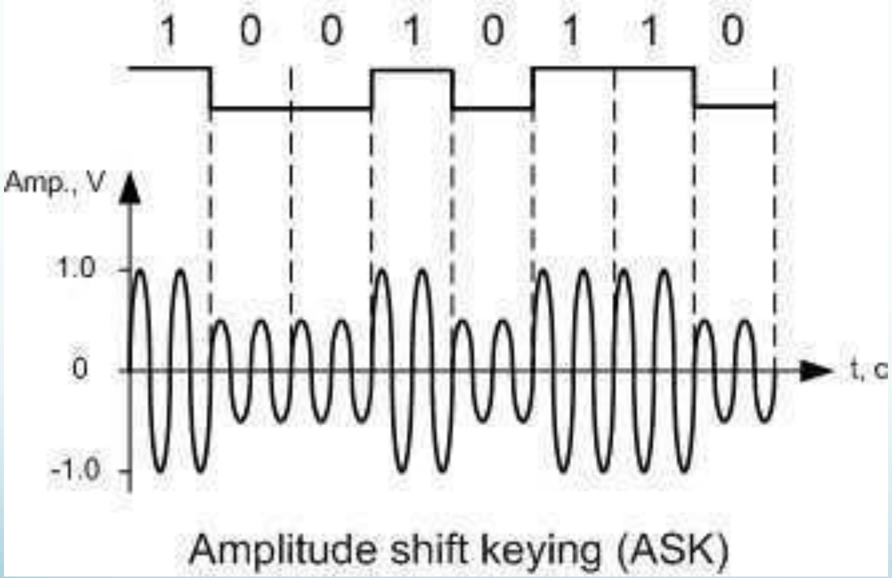
RF Modules

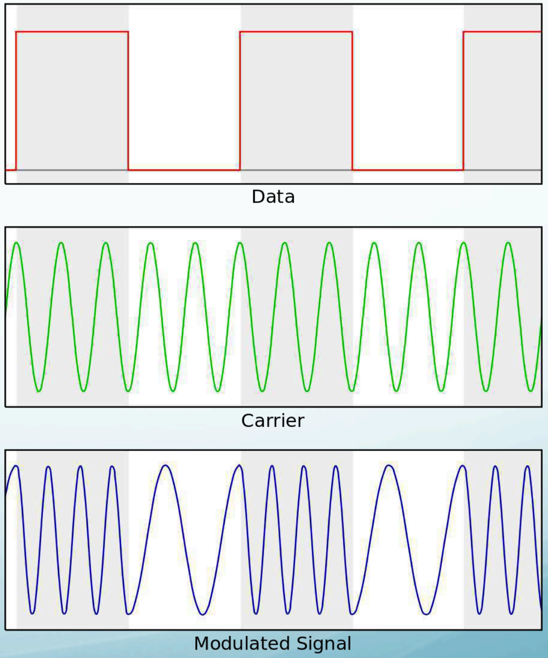
- Communicate wirelessly
- Types:
  - Transmitter module
  - Receiver module
  - Transceiver module
  - System on chip module



Modulation


S
impulse generator
$H_t(f)$
s(t)
$H_c(f)$
n(t)
$H_r(f)$
z(t)
A/D
z[k]
transmitter
channel
receiver

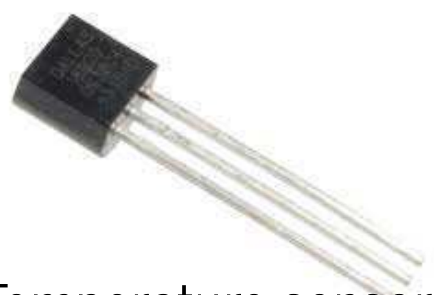

1 0 0 1 0 1 1 0
Amp., V
1.0
0
-1.0
t, c
Amplitude shift keying (ASK)


Data
Carrier
Modulated Signal

Wireless Protocol

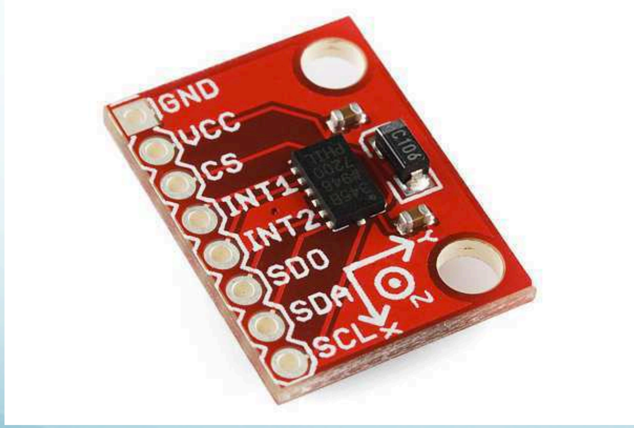
- Wifi
- Bluetooth
- Zigbee
- …

Sensors



Temperature sensor

light sensor

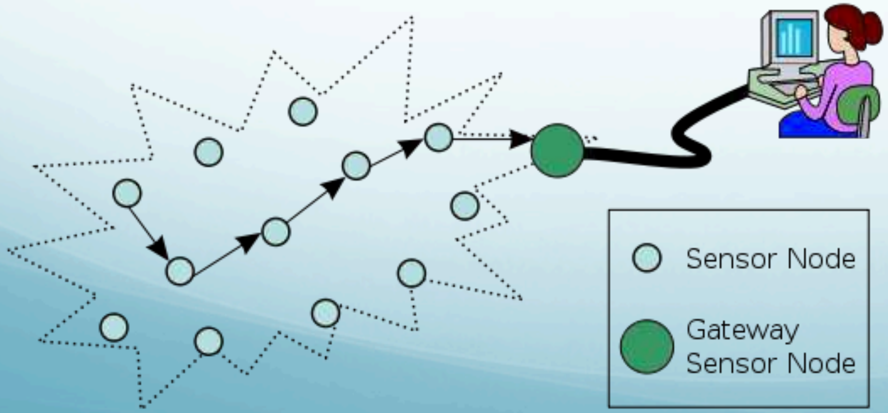


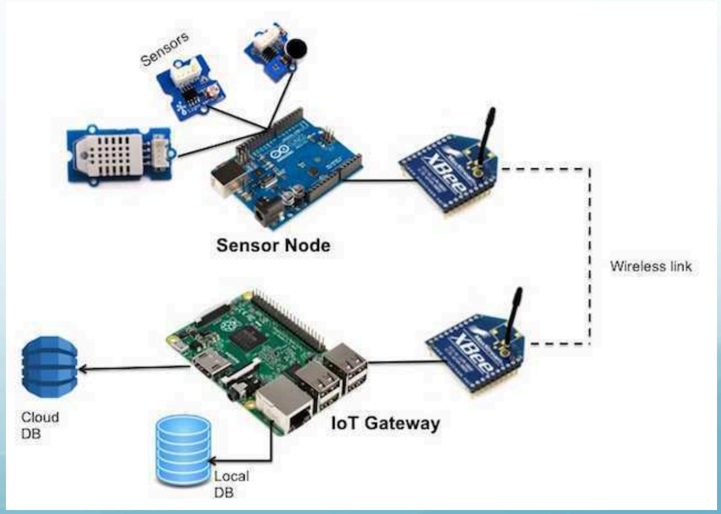
Accelerometer sensor

Temperature and humidity sensor

Wireless Sensor Network

- Spatially distributed automated sensors
  - Sensor node
  - Base station
- Applications





Thank you!

Reference

- 1. Xbee UART Data Flow Graph: https://www.digi.com/resources/documentation/digidocs/pdfs/90002002.pdf
- 2. ASK: http://www.tmatlantic.com/encyclopedia/index.php?ELEMENT_ID=10420
- 3. FSK: https://en.wikipedia.org/wiki/Frequency-shift_keying
- 4. ASK procedure: https://en.wikipedia.org/wiki/Amplitude-shift_keying
- 5. Wifi graph: https://www.lifewire.com/guide-to-wireless-network-protocols-817966
- 6. Bluetooth graph: https://en.wikipedia.org/wiki/Bluetooth_low_energy
- 7. Zigbee graph: http://buildyoursmarthome.co/home-automation/protocols/zigbee/
- 8.Temperature sensor: https://solarbotics.com/product/35040/
- 9. light sensor: https://www.intorobotics.com/common-budgeted-arduino-light-sensors/
- 10. Accelerometer sensor: https://learn.sparkfun.com/tutorials/accelerometer-basics
- 11. Temperature and humidity sensor: http://www.ebay.com/itm/DHT11-Temperature-and-Humidity-Sensor-Module-for-Arduino-/271096647277
- 12. Wireless sensor network architecture: https://en.wikipedia.org/wiki/Wireless_sensor_network#/media/File:WSN.svg
- 13. Sample for WSN: https://thenewstack.io/tutorial-prototyping-a-sensor-node-and-iot-gateway-with-arduino-and-raspberry-pi-part-1/

Embedded Systems in Athletic Training

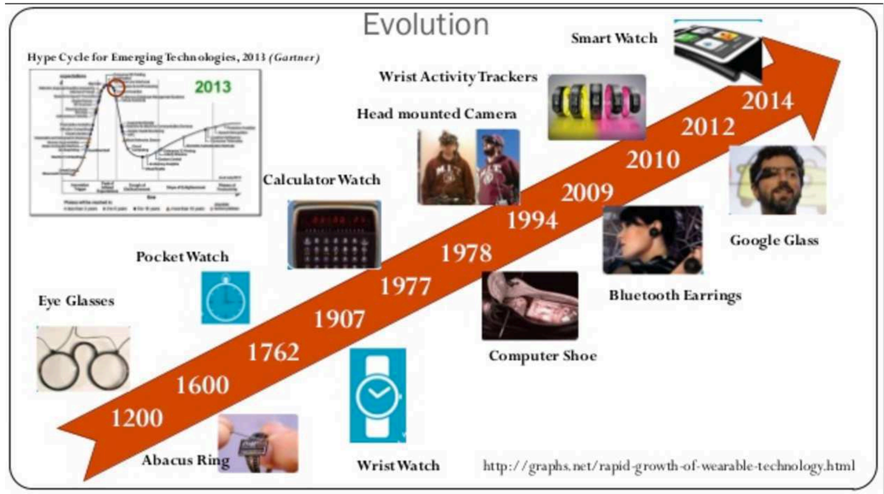
By: John Maxey

March 28, 2016

EECS 373

Evolution of Training Technology

- Research and development constantly changing the way athletes train
- Innovations in:
  - Apparel – clothing (heatgear), shoes
  - Equipment – tennis racket, bicycle
  - Biometrics – pedometers, HR monitors
  - Mobile apps – AMP Sports
  - Wearable devices – FitBit, motusPRO
- Wearable technology is a $14 billion industry



Benefits of Technology in Athletic Training

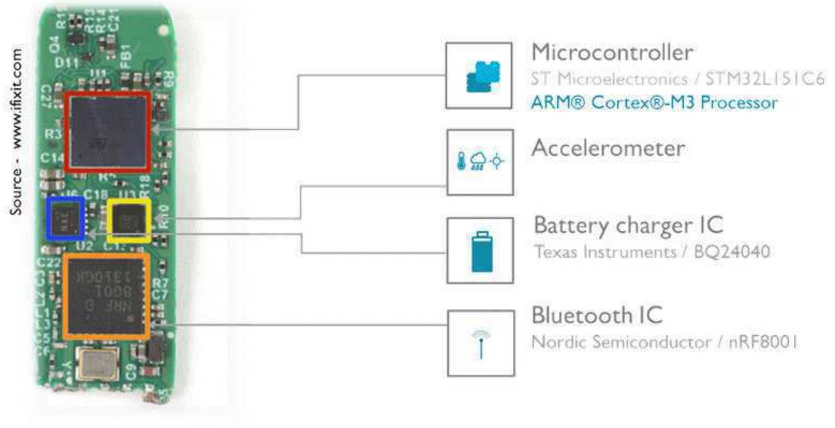
- Analyzes data in real time
  - Heart rate
  - Calories
  - Distance
  - Steps
- Understand body’s reactions during training
  - Comparable to a dashboard on a car
- Continues to get smaller, more powerful, and cheaper

Aspects of Training Technology

- Water tolerance – able to withstand sweat while training
- Size – must not interfere with performance
- Power consumption – must conserve battery life to last long enough
- Wireless communication – connect with other devices to display data
- Microcontroller – determine the capabilities of the device

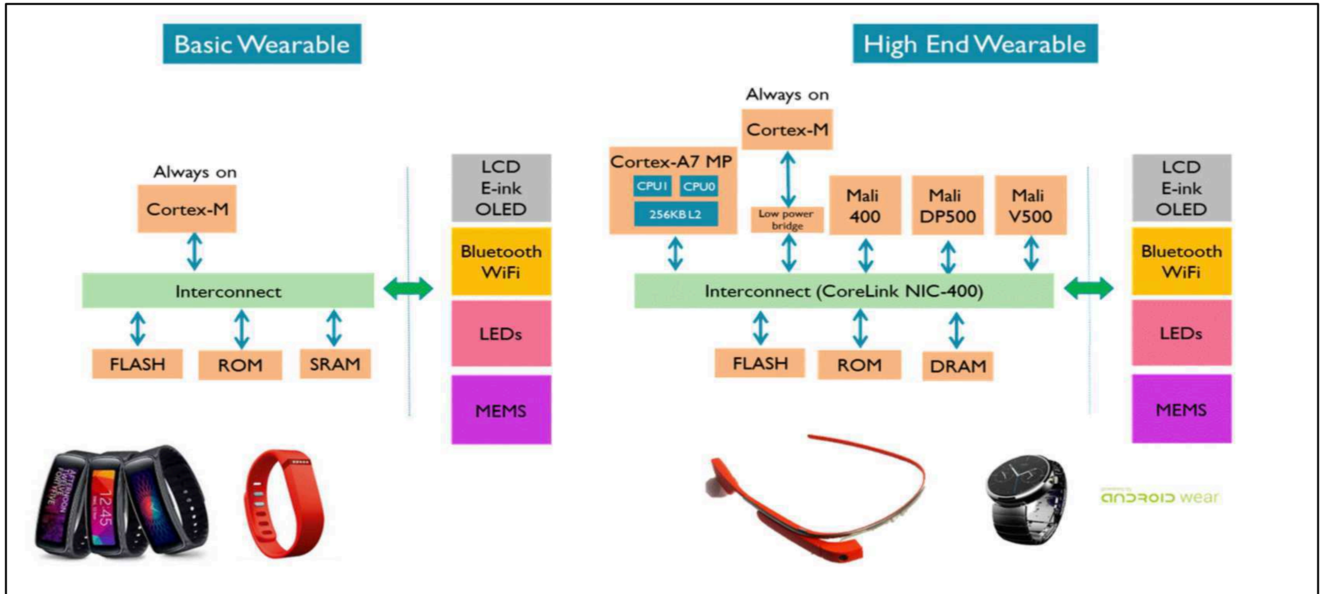
How it Works

- Processor always on – motions/activity trigger interrupts
- RTOS – real-time operating system, processes data without buffers
- ARM processor: interfaces with sensors and RFID,
  displays data on LCD screen
- Bluetooth: link to smartphone



How it Works (continued)

- Accelerometer, pedometer, HR monitor, etc. tracks activity
- Data points from sensors estimate current state
- High-end products: multiple processors, connect to cloud services, user interface provides smartphone graphics , advanced operating system



Basic wearable to high end wearable

Advanced Training and Analysis - motusPRO

- Used to track exact motions of baseball players
- Tracks over 40 mechanical metrics
- Assists in technique, trends, and rehabilitation
- Small, lightweight sensors in clothing
- Transmits data to app in real-time
- CAD advancements allow for
  virtual design and testing

References

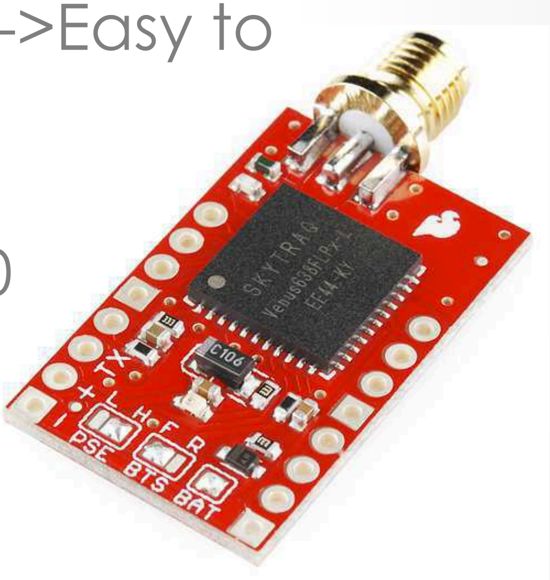
- http://www.dailymail.co.uk/sciencetech/article-2138142/Electric-training-suit-vibrates-tell-Olympic-athletes-perfected-routine.html - electric training suit
- https://www.forbes.com/sites/paullamkin/2016/02/17/wearable-tech-market-to-be-worth-34-billion-by-2020/#74051eb13cb5 - wearable technology market
- http://www.motusglobal.com/motuspro.html - motusPRO
- https://community.arm.com/iot/embedded/b/embedded-blog/posts/arm-technology-driving-the-wearable-trend - ARM technology in sports
- https://www.slideshare.net/Funk98/wearable-technology-design - evolution of sports technology
- http://www.embedded.com/design/real-world-applications/4431259/The-basics-of-designing-wearable-electronics-with-microcontrollers - designing wearable technology

Positioning Methods in Embedded Systems

Jacob Cooper

GPS

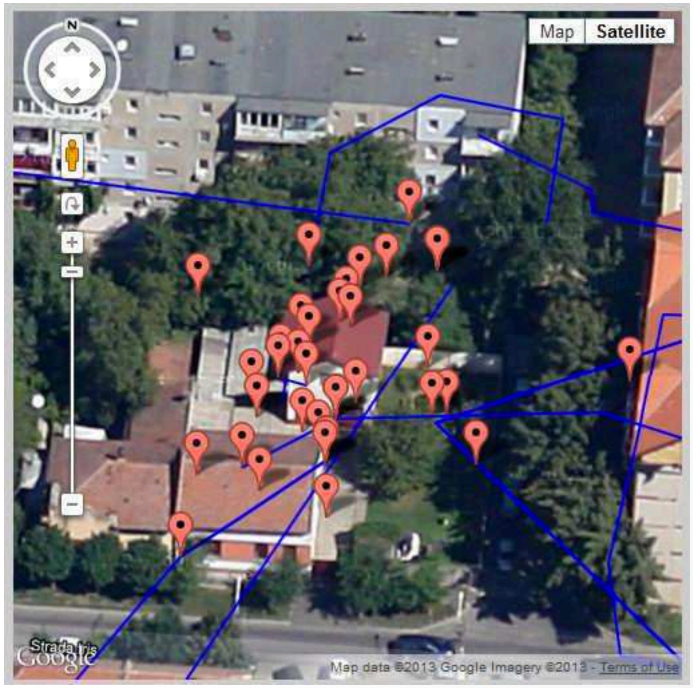
- Tracking via satellites
- Works globally
- Very commonplace(smartphones) ->Easy to implement into system
- GPS modules on sparkfun for $40-80



GPS

CONS

- Inconsistent accuracy(smartphone GPS 16ft)
- Ineffective indoors
- Mildly power hungry



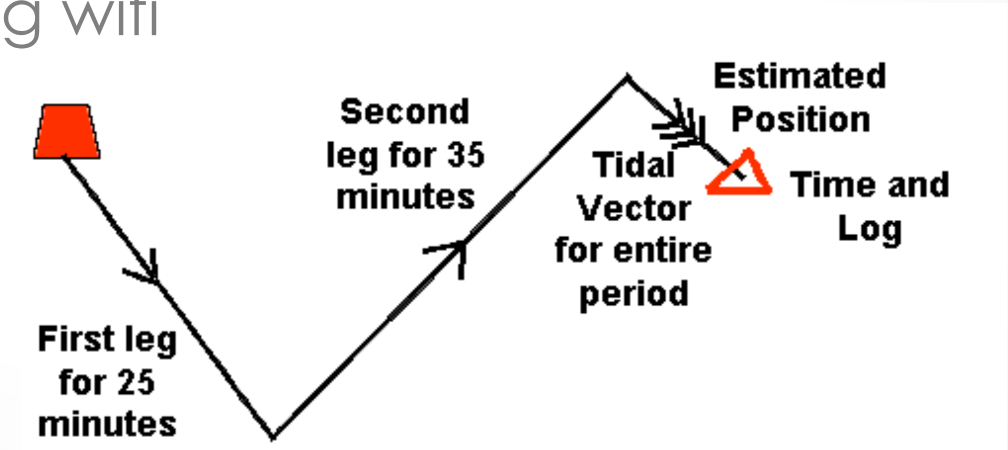
Wifi Based Positioning

- Calculate using strength of wifi signal from access points with known locations
- Good solution for indoor locations with wifi
- Arduino function wifi.rssi()

- Limited settings
- Median accuracy of 2-4m

Dead Reckoning

- Use initial position and movement calculations
- IMU is good solution(sparkfun $14-50)
- Pairs with GPS tracking for indoors
- Cumulative error builds up
- Reset/refresh using wifi



Ultrasonic

- Works locally, requires line of sight
- One side transmits and one receives
- Direction and distance applications
- Cheap, low power options
- Consider echoing effects

Infrared

- Local, requires line of sight
- Single ended
- Cheap options work within 5ft
- Affected by conditions especially lighting
-

Lidar

- Expensive($1,000's) for sweeping
- Cheaper option($150)
- Near-infrared laser
- 40m Range, 2.5cm accuracy
- Setup for I^2C or PWM

Questions?

AUDIO PROCESSING IN EMBEDDED SYSTEMS

BY THEO MILLER

AUDIO SAMPLING

- According to the Shannon-Nyquist Theorem, properly reconstructing a signal requires sampling at twice its frequency
- Range of human hearing is 20Hz – 20kHz, so sampling rate must be at least 40 kHz
- Low-pass filter needed, lowering effective sample rate
- Standard Digital audio samples at 44.1 kHz to compensate
- Sample rates of 48, 96, or 192 kHz also exist, but there is much debate as to whether they increase quality
- Other systems, such as voice recognition and reproduction, use lower sample rates, as most of the higher frequencies aren't needed

AUDIO OUTPUT FORMATS

- Pulse Code Modulation (PCM)
  - Most common
  - Amplitude of sample represented as digital code
  - Used by most standard ADC's and DAC's
- Pulse Width Modulation (PWM)
  - Amplitude encoded in duty cycle
  - Requires PWM carrier frequency to be at least 12 times the bandwidth of the signal
  - Speakers require filter to remove carrier frequency
  - Does not require a DAC, can be sent from GPIO or specialized PWM output
  - Low cost

AUDIO OUTPUT FORMATS

- Direct Stream Digital (DSD)
  - Developed by Sony and Phillips
  - Most modern ADC’s and DAC’s use sigma-delta designs
  - Involves over sampling signal at 1 bit data resolution
  - PCM requires extra conversions
  - Less intuitive to process than PCM, requires extra overhead

AUDIO COMPRESSION

- Uncompressed audio takes up large amount of space
  - CD-quality audio, ~10MB for 1 min
  - Examples: WAV and AIFF
- Lossless Compression
  - Reduces size by ~½, bit-perfect copy
  - Examples: FLAC, ALAC, APE
- Lossy Compression
  - Can reduce size by 10x or more, information lost
  - Uses quirks in ear's physiology to remove data without drastically affecting audio quality
  - Examples: MP3, AAC, WMA

AUDIO CODECS

- Integrate ADC’s, DAC’s, and audio compression into one system
- Usually support a wide rage of communication protocols
- Highly configurable

SOURCES

- http://www.analog.com/media/en/dsp-documentation/embedded-media-processing/embedded-media-processing-chapter5.pdf
- http://www.trustmeimascientist.com/2013/02/04/the-science-of-sample-rates-when-higher-is-better-and-when-it-isnt/
- http://lifehacker.com/5927052/whats-the-difference-between-all-these-audio-formats-and-which-one-should-i-use

QUESTIONS?

REAL TIME OPERATING SYSTEM

Yi Zhi Wee

EECS 373

WHAT IS RTOS?

- OS for applications with real-time constraints
- Must respond to events quickly
- No deadloop
- Provides library for task scheduling

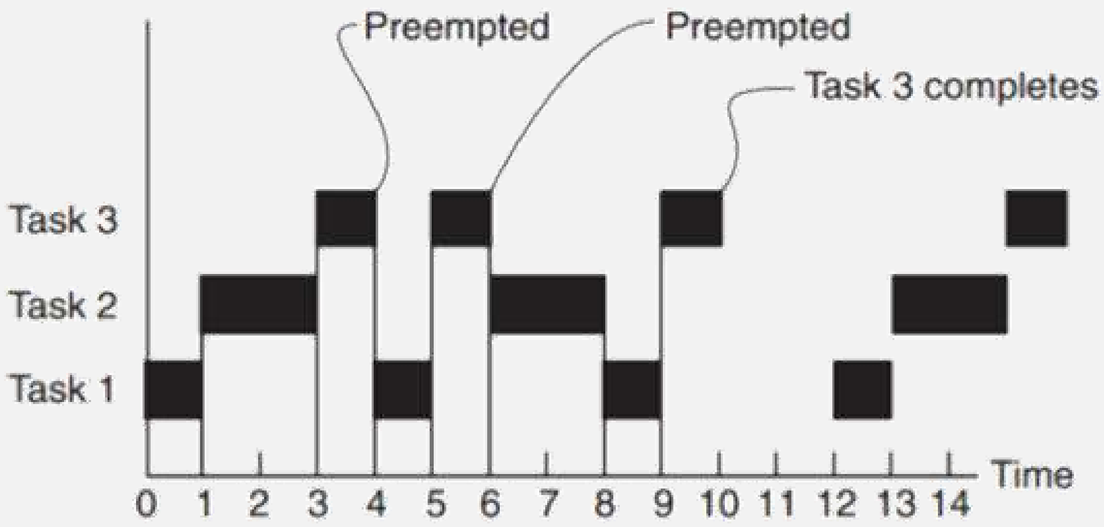
JUST USE NVIC?

- Must manually setup hardware
- Scheduling will (probably) need timer
- RTOS schedules task in software (easier debugging)
- Program portable to other machines with same RTOS

EXAMPLE: RATE MONOTONIC SCHEDULING

- Static priority
- Tasks are periodic
- Shortest period highest priority

Task	Execution Time	Period	Priority
T1	1	4	High
T2	2	6	Medium
T3	3	12	Low



WHAT ELSE?

- Dynamic priority
- Interrupts (low latency)
- Other scheduling algorithms (eg. round robin)

Random Numbers in Embedded Systems

Brennan Garrett

Applications of Random Numbers

- Cryptography (random keys)
- Network Applications
- Games

Software Generators vs Hardware Generators

Software	Hardware
C Function: Rand()	Input Time Differences
Kiss Algorithm	Noise from ADC

Rand()

In 1972 C

```
static unsigned long int next = 1;

int rand(void) // RAND_MAX assumed to be 32767
{
        static const unsigned long int a = 1103515245;
        static const unsigned short b = 12345;
        next = next * a + b;
        return (unsigned int)(next/65536) % 32768;
}

void srand(unsigned int seed)
{
        next = seed;
}
```

https://www.slideshare.net/numericalsolution/random-number-generation-in-c-past-present-and-potential-future

- Very easy to use, no implementation necessary
- Poor quality of randomness, produces cyclic results for lower numbers
- Poor randomness, useful for trivial applications

Kiss Algorithm

```
def uint32(i):
    return i & 0xFFFFFFFF

def kiss():
    # LCG:
    x = uint32( 69069 * x + 12345 )

    # Xorshift
    y ^= uint32(y << 13)
    y ^= uint32(y >> 17)
    y ^= uint32(y << 5)

    # Multiply-with-carry
    t = 698769069 * z + c;
    c = uint32(t >> 32)
    z = uint32(t)

    # Combining all 3
    return uint32(x + y + z)
```

https://www.embedded-office.com/en/blog/random-1.html

- Keep it Simple Stupid
- Multiple-With-Carry Generator, Shift Registers, Linear Congruential Generator
- Provides better "randomness"
- Better software implementation, not perfect

Input Time Differences

```
static void keypress_seed_init()
{
        /* Clear all keypresses first. */
        while (button_tstc())
                button_getc();

        /* Wait for a key. */
        button_getc();

        srand(systick_get_ticks());
}
```

www.zilogic.com/blog/tutorial-random-numbers.html

- Measures time between two input signals (keyboard, button)
- Time difference provides random seed
- Can be implemented at start time

Noise from ADC

```c
static void adc_seed_init()
{
        int i;
        int seed;
        unsigned lsb;

        adc_enable(1);

        /* Collect the LSB bits of 32 consecutive samples. */
        seed = 0;
        for (i = 0; i < 32; i++) {
                lsb = adc_read16(1);
                seed |= (lsb << i);
        }

        srand(seed);
}
```

http://www.zilogic.com/blog/tutorial-random-numbers.html

- Application reads in thermal noise from an ADC
- This physical measurement provides pure randomness
- Best method to find random seed

Conclusion

- Measuring a physical phenomena as a seed will produce the best results
- Randomness relates to application

References

- http://www.azillionmonkeys.com/qed/random.html
- http://www.embedded.com/design/configurable-systems/4024972/Generating-random-numbers
- http://www.zilogic.com/blog/tutorial-random-numbers.html

Interfacing with N64 Controller

James Mitchel

The Controller

- Controller for N64
- First to utilized analog stick for 3D gameplay
- 14 buttons and analog stick for control
- Trident shape still unique today

http://how-does-things-work.blogspot.com/2010/01/working-of-nintendo-64.html

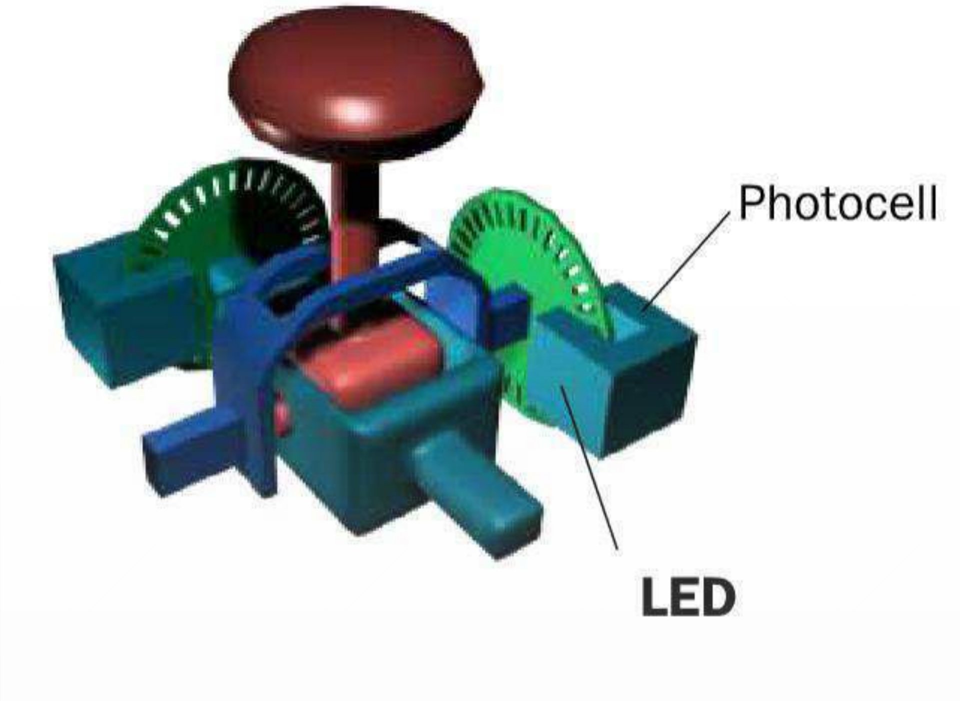
The Buttons

- Each button is a switch that completes a circuit when it is pressed

http://www.neogaf.com/forum/showthread.php?t=1181939

The Joystick

- Two wheels, with tiny slots around the edge, form right angle
- Moving the joystick moves the two wheels turn slightly
- Wheels in between LED and photo cell
- Quadrature encoding!



http://how-does-things-work.blogspot.com/2010/01/working-of-nintendo-64.html

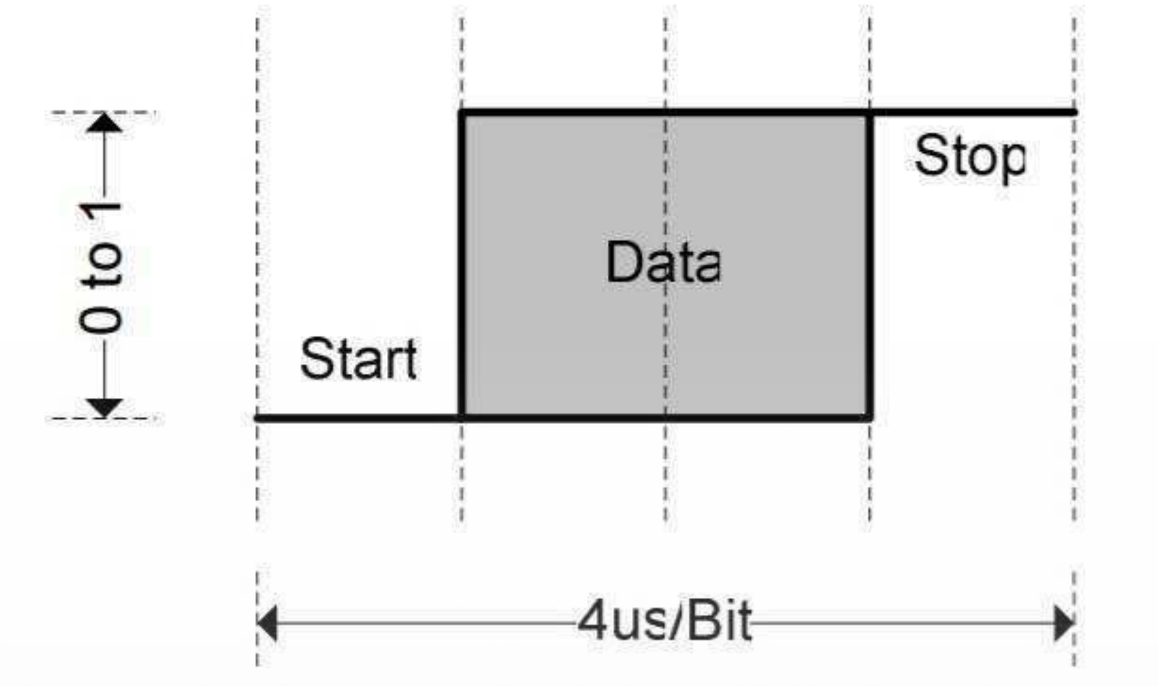
The Serial Port

- One wire for power (3.3 V) and one for ground
- Only one wire for data
- Open collector
- Needs own serial interface
- Self clock

http://how-does-things-work.blogspot.com/2010/01/working-of-nintendo-64.html

The Bit

- Self clocking
- Each bit lasts 4us
- Starts low
- Ends high
- Data is the middle
- 0 when low
- 1 when high



http://www-inst.eecs.berkeley.edu/~cs150/fa04/Lab/Checkpoint1.PDF

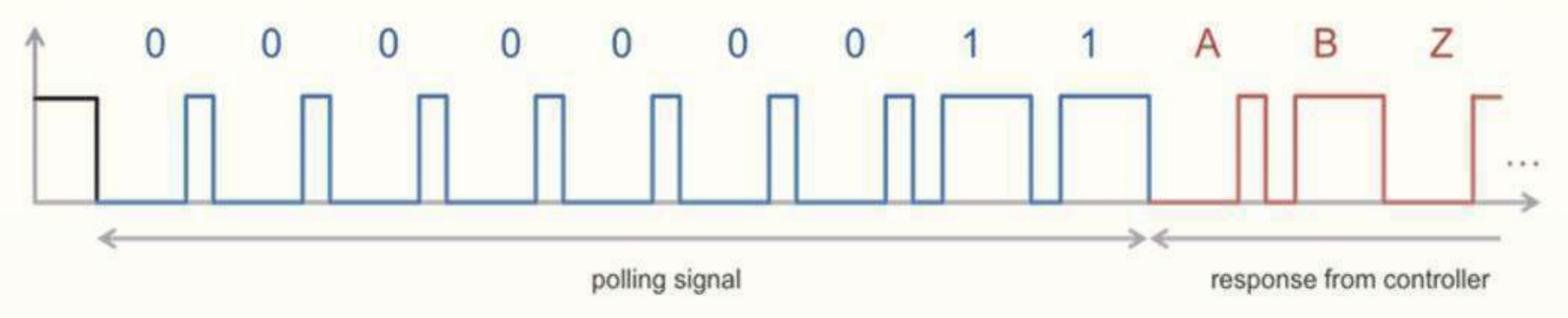
The Commands

- 8’hFF: Reset Controller
- 8’h00: Get Status
- **8’h01: Get Buttons**
- 8’h02: Read Mempack
- 8’h03: Write Mempack
- 8’h04: Read EEPROM
- 8’h05: Write EEPROM

http://how-does-things-work.blogspot.com/2010/01/working-of-nintendo-64.html

Polling

- Send message to controller over data wire
- The message is a byte long plus a stop bit (so effectively 9 bits)
- Message is 0x01 for getting button data
- So send 0b000000011 over the data line using the bits described before



http://www.pieter-jan.com/node/10

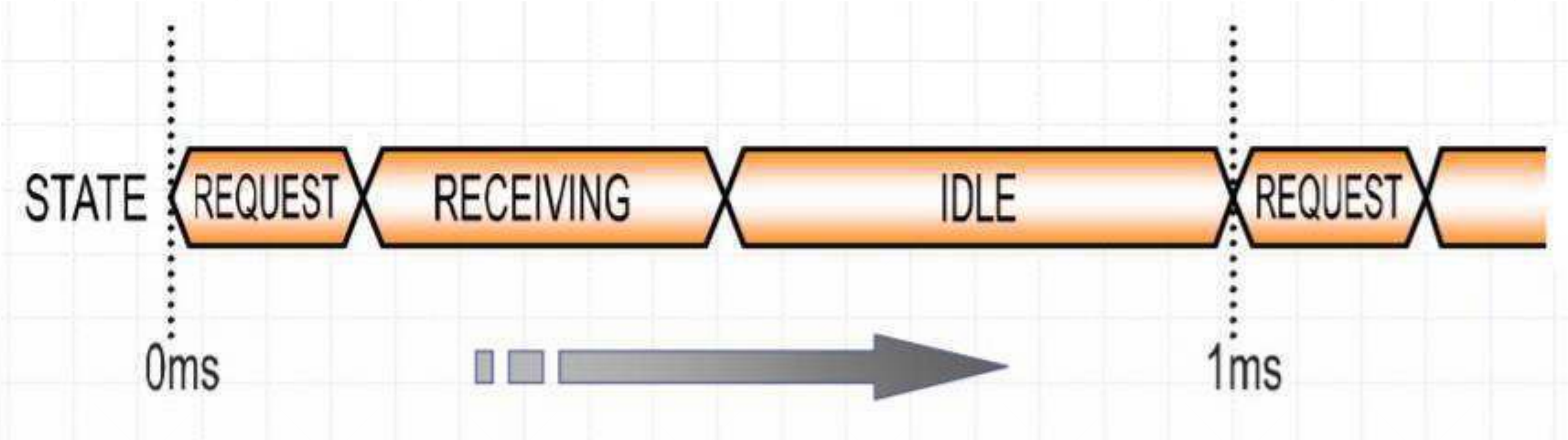
Button Status

- Controller responds over data wire
- Sends 4 bytes plus a stop bit (so effectively 33 bits)
- Buttons sent as binary, pressed versus not pressed
- Receive joystick x-coordinate and y coordinate

Byte	Data[7]	Data[6]	Data[5]	Data[4]	Data[3]	Data[2]	Data[1]	Data[0]
1	A	B	Z	Start	D-Up	D-Down	D-Left	D-Right
2	Joystick Reset	0	L	R	C-Up	C-Down	C-Left	C-Right
3	Signed joystick x-axis coordinate							
4	Signed joystick y-axis coordinate							

http://slideplayer.com/slide/8085899/

Sending and Receiving



http://www-inst.eecs.berkeley.edu/~cs150/sp01/Labs/lablecckpt1.pdf

The Interface

- Most material online don’t use a FPGA so they are polling and receiving the data all from software.
- For my team’s application it makes more sense to use FPGA and interrupts to interface between the controller.
- Have the FPGA constantly poll, get button data, and send an interrupt when an important event (like button press) happens so software can react.

References

- http://www-inst.eecs.berkeley.edu/~cs150/fa04/Lab/Checkpoint1.PDF
- http://www-inst.eecs.berkeley.edu/~cs150/sp01/Labs/lablecckpt1.pdf
- http://www.pieter-jan.com/node/10
- http://how-does-things-work.blogspot.com/2010/01/working-of-nintendo-64.html
- https://www.eecs.umich.edu/courses/eecs270/lectures/270L23NotesF14.pdf
- http://slideplayer.com/slide/8085899/
- http://www.neogaf.com/forum/showthread.php?t=1181939

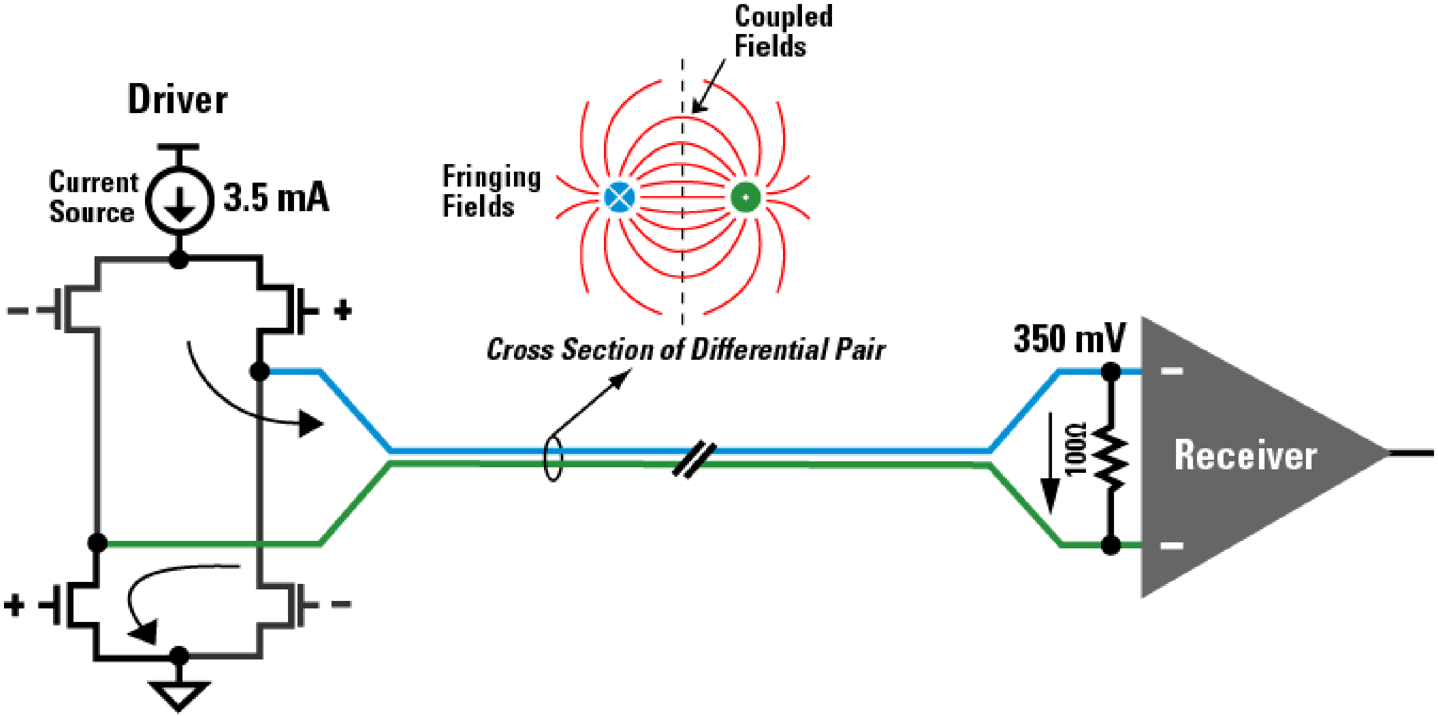
LVDS I/O Standard

By: Jacob Sigler

What is LVDS?

- Stands for Low Voltage Differential Signaling
- Transmits inverted and non-inverted signal called "Differential Pair" or "Diff Pair"
  - Signals measured between each other, not ground reference
- Voltage swing of ~±350mV (compared to 3.3V for CMOS logic)
- Max data rate ~3.125 Gbps

LVDS Driver



*1

LVDS Advantages – Low Power

- Lower voltage results in lower dynamic power
- For our FPGA:

Power per I/O Pin

Table 2-10 • Summary of I/O Input Buffer Power (per pin) – Default I/O Software Settings Applicable to FPGA I/O Banks, I/O Assigned to EMC I/O Pins

	VCCFPGAIOBx (V)	Static Power PDC7 (mW)	Dynamic Power PAC9 (µW/MHz)
Single-Ended			
3.3 V LVTTL / 3.3 V LVCMOS	3.3	–	17.55
2.5 V LVCMOS	2.5	–	5.97
1.8 V LVCMOS	1.8	–	2.88
1.5 V LVCMOS (JESD8-11)	1.5	–	2.33
3.3 V PCI	3.3	–	19.21
3.3 V PCI-X	3.3	–	19.21
Differential			
LVDS	2.5	2.26	0.82
LVPECL	3.3	5.72	1.16

- For 200MHz Signal: LVDS = 2588uW 3.3v LVTTL = 3510uW

LVDS Advantages – High Speed

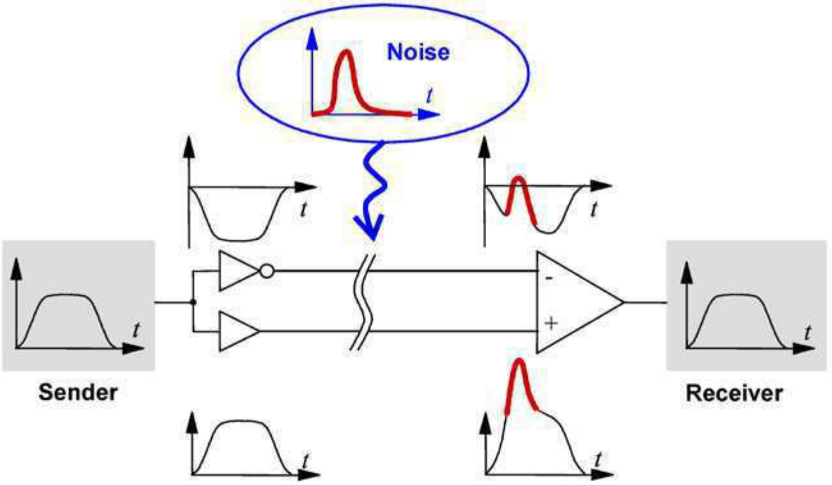
- Low voltage swing also allows higher speed (charging capacitance)
- For our FPGA:

***Table 4-5* • Maximum I/O Frequency for Single-Ended and Differential I/Os (maximum drive strength and high slew rate selected)**

Specification	Performance Up To
LVTTL/LVCMOS 3.3 V	200 MHz
LVCMOS 2.5 V	250 MHz
LVCMOS 1.8 V	200 MHz
LVCMOS 1.5 V	130 MHz
PCI	200 MHz
PCI-X	200 MHz
LVDS	350 MHz
LVPECL	300 MHz

LVDS Advantages – Noise Immunity

- Common mode noise couples equally into both signal lines
- Receiver takes difference of inputs, so common mode noise is subtracted out



*2

LVDS Disadvantages – Two Lines

- LVDS requires inverted and non-inverted signals, so two wires per line
- To get around this, can run ½ lines at 2x speed compared to parallel interface
  - 12 parallel lines at 100MHz
  - 6 LVDS pairs at 200MHz

LVDS Uses

- Common Embedded Uses:
  - LCD Video Connectors
  - Camera Interface
  - High Speed ADC/DAC Interface

Implementation Tips

- SmartFusion has pre paired diff inputs
  - Cant route two random signals and call them diff pairs
- Can select LVDS in the IO manager

References

1. By Dave at ti - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=19127216
2. By Linear77 - Own work, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=18321195

PID Control in Embedded Systems

Shaurav Adhikari

EECS 373

Example application of PID control in an Embedded System

Controlling the position of an actuator by getting its current position as feedback

Option P – Potentiometer Position Feedback

WIRING: (see last page for pin numbering)

1 - Orange – Feedback Potentiometer negative reference rail

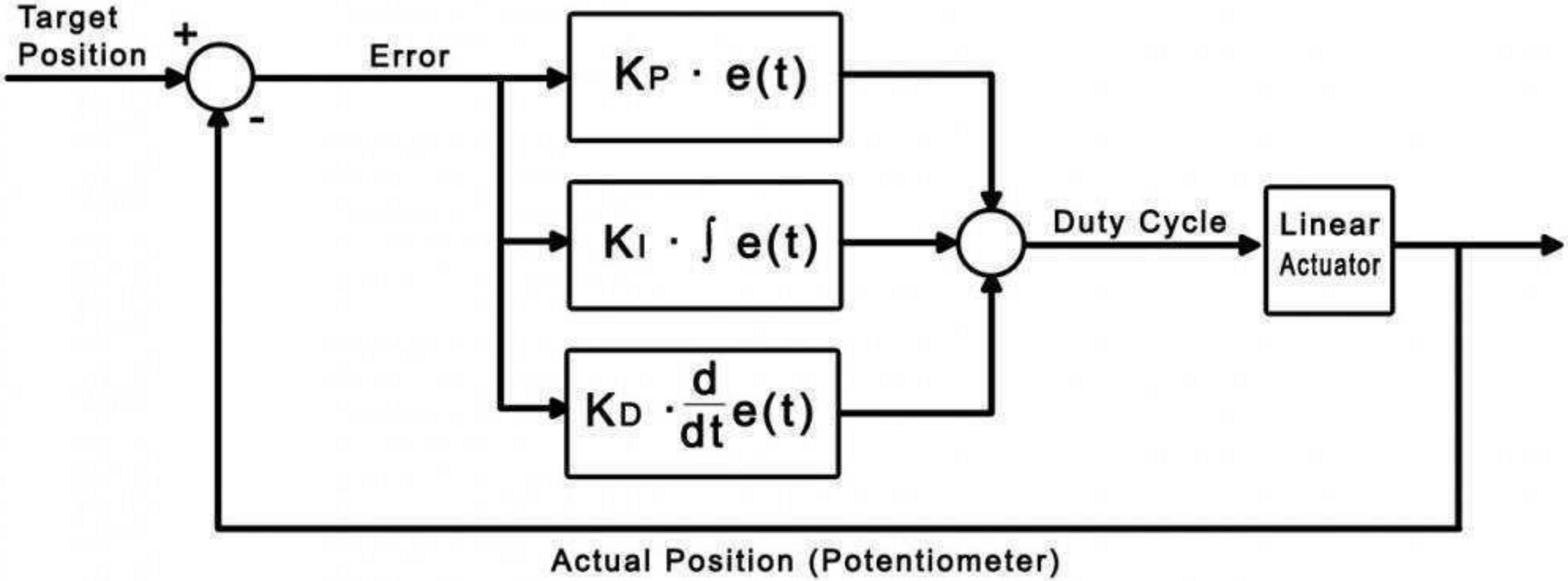
2 - Purple – Feedback Potentiometer wiper

3 - Red – Motor V+ (12V)

4 - Black – Motor V- (Ground)

5 - Yellow – Feedback Potentiometer positive reference rail

PID controllers use feedback to determine the output



PID controller

Proportional:

- Quickly moves output in the desired direction and reverses if overshooting occurs

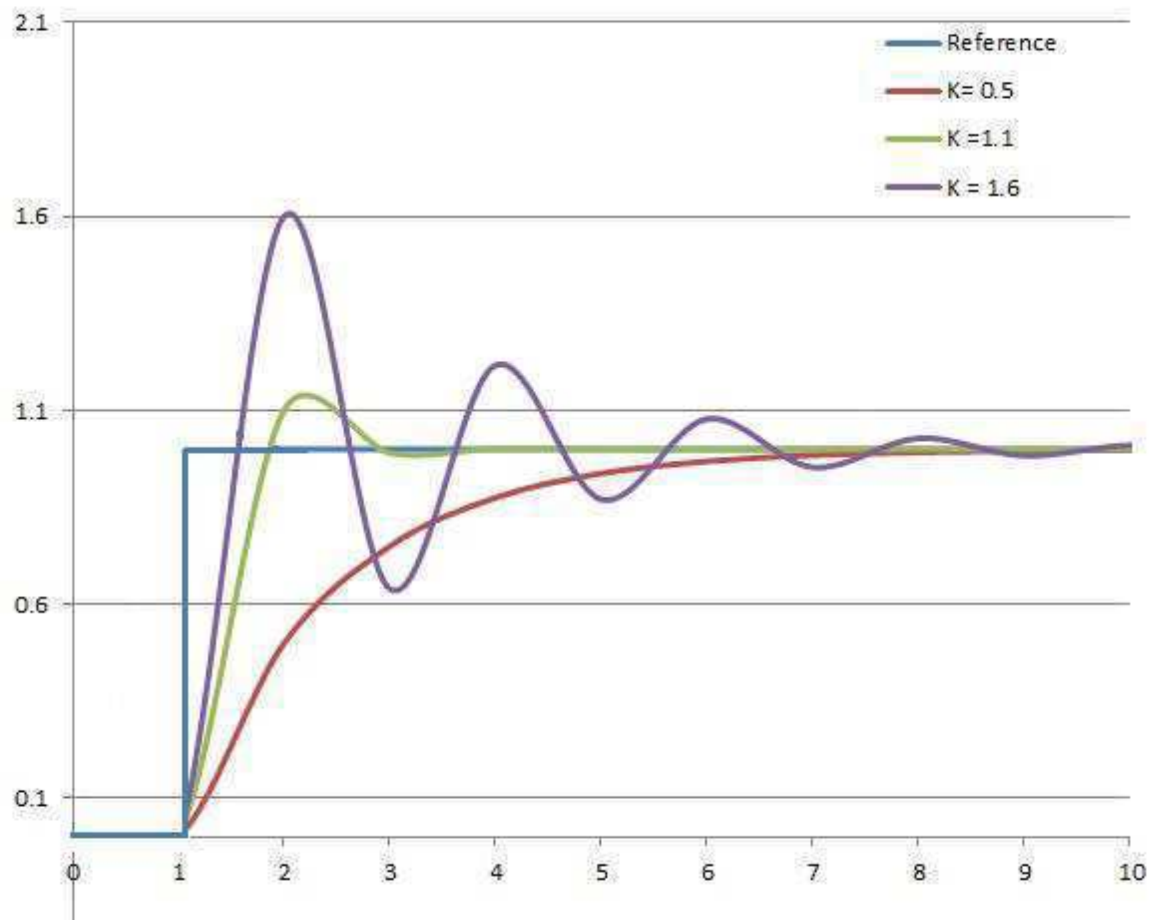
Integral:

- Corrects small steady state errors by accumulating them over time.

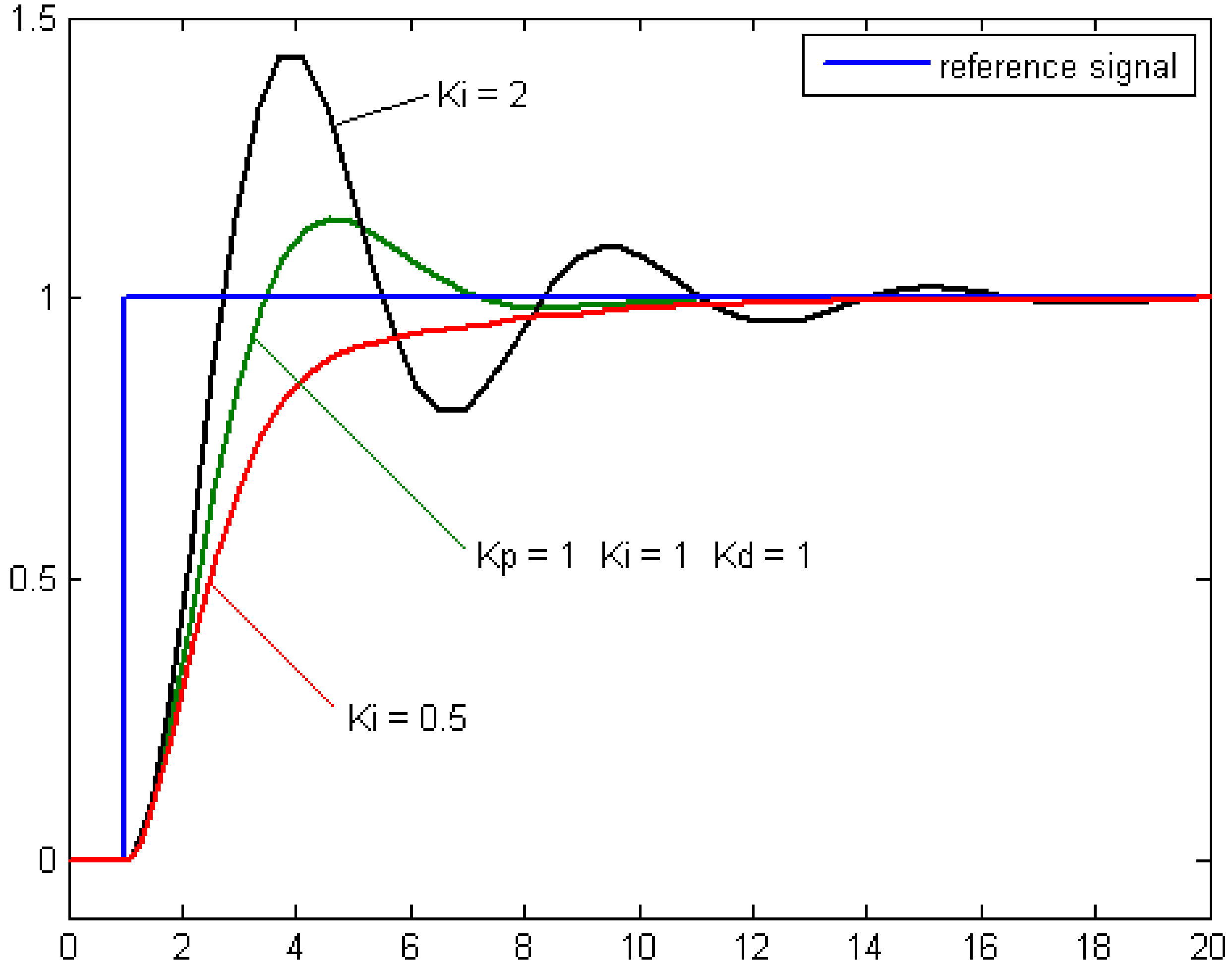
Derivative:

- Allows for higher Kp and Ki values without overshooting.
- Limits how quickly changes occur in output.

Varying Kp

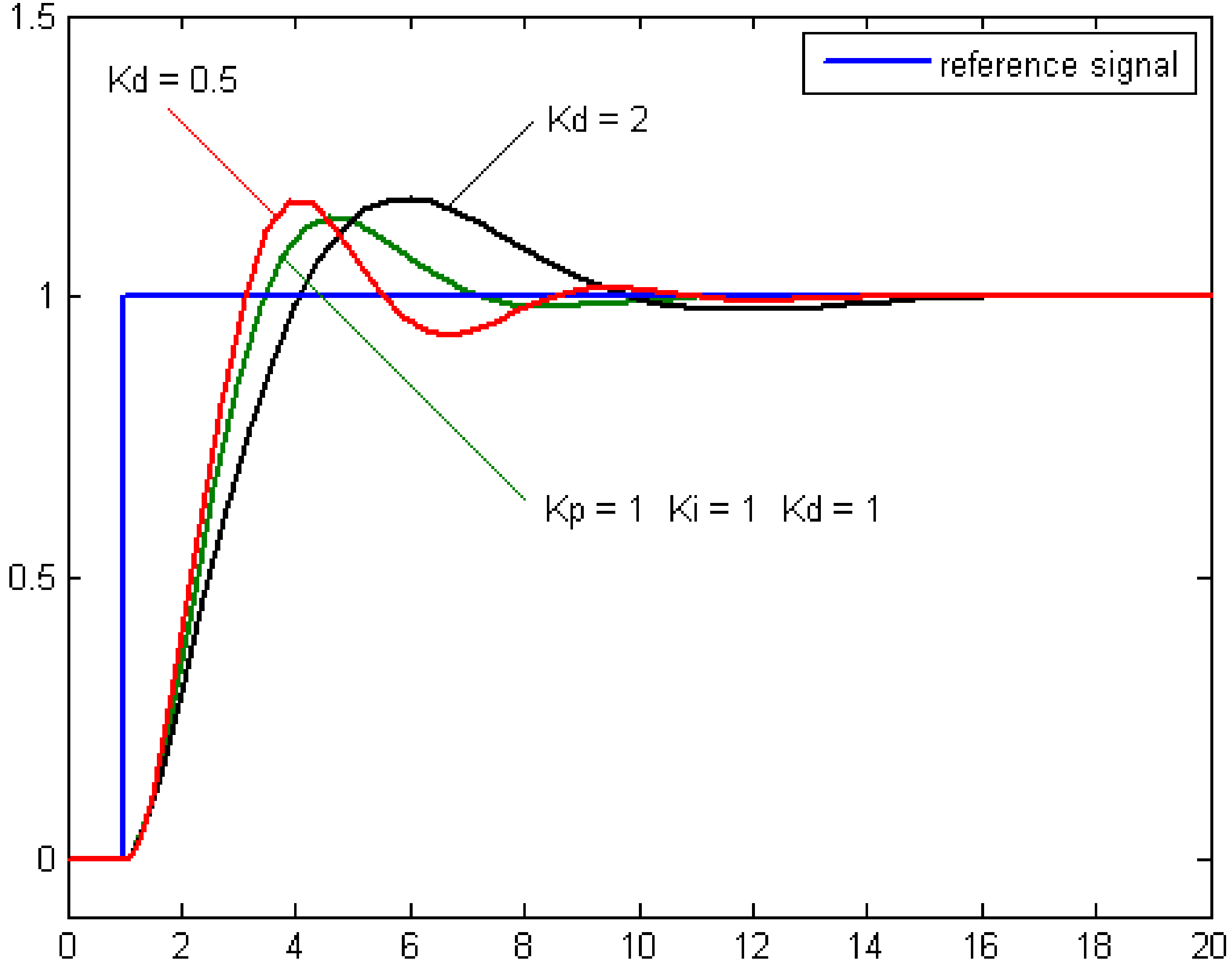


Varying Ki



1.5
1
0.5
0
0
2
4
6
8
10
12
14
16
18
20
reference signal
Ki = 2
Kp = 1 Ki = 1 Kd = 1
Ki = 0.5

Varying Kd



Kd = 0.5
Kd = 2
Kp = 1 Ki = 1 Kd = 1
reference signal
1.5
1
0.5
0
0
2
4
6
8
10
12
14
16
18
20

Implementing PID

Loop:

current_error = desired_position – current_position

proportional = Kp * current_error

integral = Ki * accumulated_error

derivative = Kd * (current_error – previous_error)

controller_output = proportional + integral + derivative

Things to consider

- If feedback is noisy then the controller would produce undesired output
- Some devices may not respond to small changes
- When error is greater than a chosen threshold, simply get the error within the threshold as fast as possible.

Questions?

References

- http://www.phidgets.com/docs/Linear_Actuator_-_PID_Control
- http://tutorial.cytron.com.my/2012/06/22/pid-for-embedded-design/
- https://en.wikipedia.org/wiki/PID_controller

Embedded System and Wearable device

JIAYI LIU

EECS 373

Basic Description

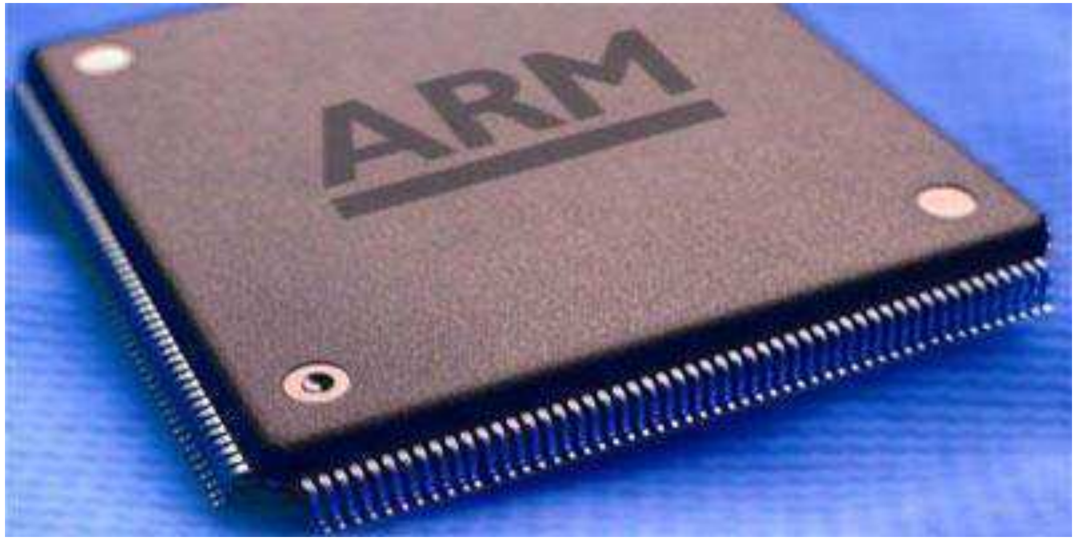
‘Wearable’ devices are miniature electronic devices worn on the body, often integrated with or designed to replace existing accessories such as a watch.

Size

- The devices must be small enough to be wearable.
- It’s always be challenging to integrate more functionalities inside a small space.
- System-on-Chip (SoC) and chip scale packages (CSP) enable engineers to minimize the size of the device.

Power Consumption

- Wearable devices need to stay on to do the monitoring while the battery capacity is limited, power consumption is very challenging.
- Solve by applying efficient algorithm(inactive unused program or functionality) or use good MCU(32-bit ARM architecture, Bluetooth Low Energy (BLE)).



Wireless communication

- Wireless communication is commonly used in wearable devices to enable devices interact with each other.

- Each device need to support at least one wireless protocols(Wi-Fi, ANT+, Bluetooth Low Energy (BLE)).

Microprocessor or Microcontroller

- The selection of the processor is highly based on features of the device. Commonly use MCUs and in most case engineers integrate functions on a single chip to minimize size.

- 32-bit ARM processors are popular in wearable devices. It's computing performance is brilliant and it's efficiency in terms of power is also ideal.

- When the system is sophisticated, multiple processor might be required. (When the system has bunch of sensors and require real-time analysis and wireless communication).

Operating system

- Not required for simple device or system.

- When the device connect with complex devices like smartphone(Android) or system itself is complex, OS may be needed.

Thank you!!

Embedded Systems in Space

Joe Lafayette
Nick Martinelli

Embedded Systems in Space

- Sophisticated embedded systems are integral to space travel
  - A spacecraft without onboard computing won't do much
- Surviving in space is exceptionally hard
  - It is an extremely hostile environment
- Even small missions require extreme preparation

Areas of Consideration

- Radiation
- Temperature
- System Reset
- Reprogramming
- Power Consumption

Radiation

- Single event upset
  - Change in device state due to a single ionizing particle
  - Single event latch-up occurs when ionizing particle short circuits device
- Bit flips
  - Can render data/instructions useless
- Europa Clipper / Multi-Flyby Mission
  - Does multiple close fly-bys to avoid intense radiation
  - Radiation would quickly render electronics inoperable

Radiation Protection

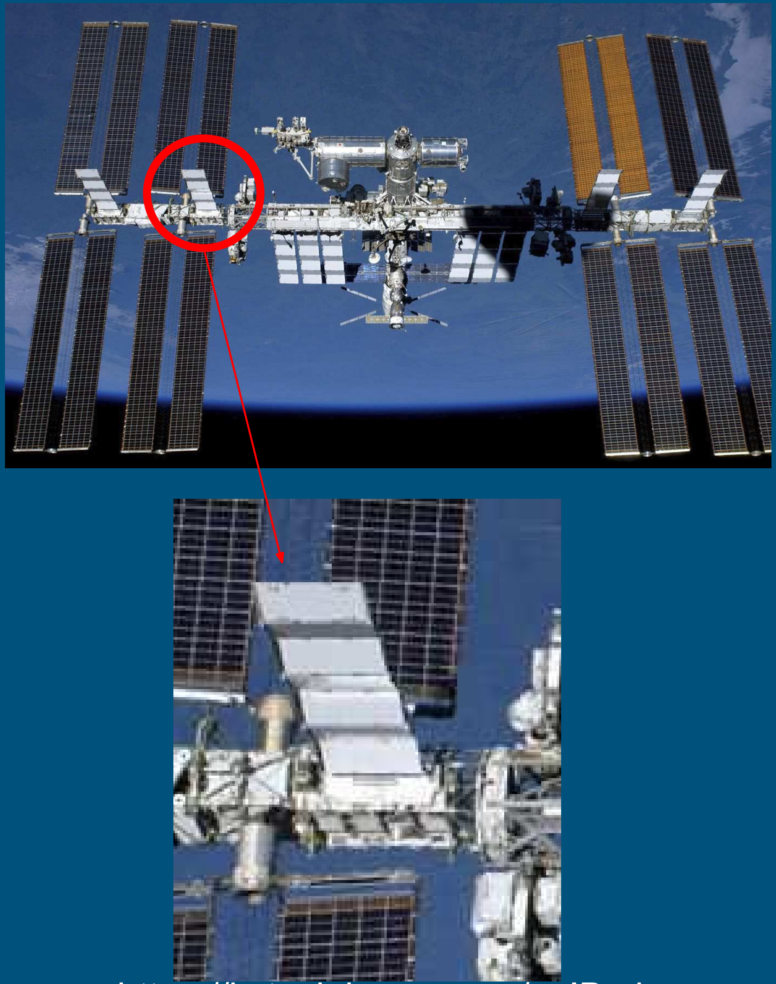
- Utilize radiation resistant/hard components
  - Can prevent bit flips and damage
- Recognize single-event latch up/upset
  - Power down components/spacecraft to minimize damage
- Shield less resistant components if needed

Temperature

- Ranges from -170º to 123º Celsius (ambient) in Low Earth Orbit
- Cold temperatures decrease the rate of chemical reactions in batteries
- High & low temperatures reduce semiconductor performance
- Components can have measurement drift as function of temperature
    - Crystal Oscillators
- You can’t get too hot or too cold

Temperature Solutions

- Heat dissipation is hard
  - Conduction and convection can’t remove heat from the system
- Can utilize thermistors to recognize low battery temperatures
  - Can activate heating circuit to keep batteries warm
- Use radiators to remove excess heat from the system
  - Requires extra mass and surface area
- Use thermal insulation to maintain temperature
- Use temperature sensors to compensate for thermal drift



https://i.stack.imgur.com/cpIBo.jpg

Temperature Solutions

- Heat dissipation is hard
  - Conduction and convection can’t remove heat from the system
- Can utilize thermistors to recognize low battery temperatures
  - Can activate heating circuit to keep batteries warm
- Use radiators to remove excess heat from the system
  - Requires extra mass and surface area
- Use thermal insulation to maintain temperature
- Use temperature sensors to compensate for thermal drift

New Horizons

https://en.wikipedia.org/wiki/New_Horizons

No Easy System Reset

- Software/hardware bugs can cause inoperable state
- Communication could be limited
  - Radio functionality may be compromised
- Need a **Watchdog Timer**
  - Can be used to reset/power cycle parts of spacecraft if not “fed”
- Well designed watchdog systems can “revive” a dead spacecraft
- Watchdog systems are invaluable
  - Lightsail 1 spacecraft
    - Didn’t have good WDT system
    - Upgrading for Lightsail 2

No Component Replacement

- Spacecraft parts can, and will, fail
  - Could mean only partial mission success
  - Could potentially interfere with operation of other components
- Can’t replace parts on a spacecraft
- Redundancy and isolation should be considered
  - Redundant copies that can be switched to in case of failure
  - Scheme for isolating faulty hardware from shared interfaces
    - Bus isolators
    - Power switching

Reprogramming

- In-flight reprogramming is essential
  - Software bugs
  - Hardware failures
- These could be fatal without reprogramming
  - Bug fixes
  - Lock-out/work around broken hardware
    - I.E., isolate from a bus, repurpose other hardware to do same job
- Don’t have a simple USB connection for reprogramming
- Need dedicated hardware/software for altering processor memory

Power Consumption

- Want system to be in sleep/low power mode whenever possible
- Need power harvesting
  - Solar
- The inverse square law is not your friend
  - Solar panels are less effective further from the sun
  - Missions to outer planets need to rely on other power sources
    - New Horizons uses a radioisotope thermoelectric generator (RTG)

New Horizons

https://en.wikipedia.org/wiki/New_Horizons

References

https://en.wikipedia.org/wiki/Single_event_upset

https://en.wikipedia.org/wiki/Europa_Clipper

https://en.wikipedia.org/wiki/Spacecraft_thermal_control

https://en.wikipedia.org/wiki/LightSail_2#LightSail_1_test_flight