Extending Android via External Microprocessors

Working outside of the box…

Mike Anderson

Chief Scientist

The PTR Group, Inc.

mike@theptrgroup.com

What We'll Talk About…

- Android and the outside world
- Strategies for adding new sensors
- Real-time Android?
- Why add external microprocessors/µCs?
- Code for the µC vs. Firmata
- Connection strategies

Android and the Outside World

- Android knows about a number of device classes out of the box
  - Gyros, accelerometers, compass, GPS, etc.
  - Integrated through libsensors into the Android framework
- Adding new sensors to the platform would normally require rebuilding the AOSP libsensors and reflashing the system
  - Works for a single platform, but it's not easily done for multiple platforms

Adding Control Capabilities

- The real world is filled with opportunities to add new interfaces
  - CAN bus, GPIOs, A/D, D/A, PWM, I2C, SPI, etc.
  - Unfortunately, it's difficult to wire these out of the typical handset/tablet
- We could build a custom Android device
  - We would need custom hardware just to wire the signals out of the Android platform
  - Additionally, there would be significant effort to get, modify and rebuild the platform sources
- Unfortunately, the Android kernel isn't tuned for even soft real-time control
  - Focus is on Java behavior

Alternate Extension Options

- Android natively supports several different connection options
  - USB, Wi-Fi, Bluetooth and NFC
- Via one of these connections, we can use an external device for the interface to the real world and use Android for control and UI
  - Offload the time-sensitive work to dedicated hardware
- Goal is to save cost while being able to guarantee service
  - We don’t need two big processors for this job

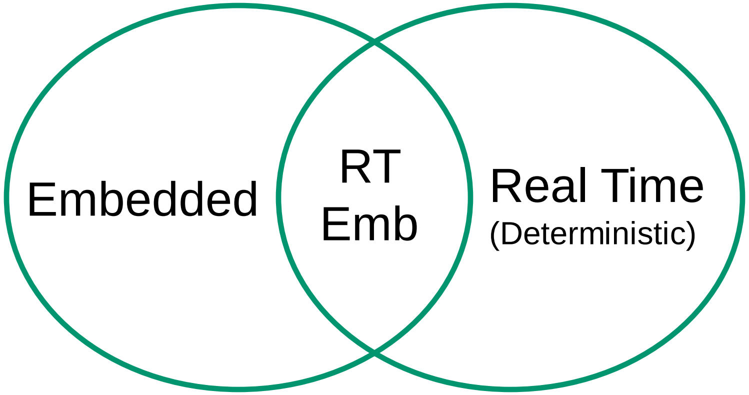
Real-Time Android?

RTDROID

- What do we mean when we say “real time”?
  - Computing with a deadline
  - The consequences for missing a deadline determine if we have “hard” or “soft” real time
- There have been many attempts to look into making Android real-time capable
  - First, we could add PREEMPT_RT to a modern kernel w/ Android support
    - But, this is just a small part of the problem
- In user space, the Dalvik VM is not even close to deterministic
    - Experiments show significant jitter and latencies
  - Replacing Dalvik is a huge undertaking and not practical

Embedded vs. Real Time

- Embedded and real time are not the same thing
  - Embedded typically means there is a computer in there someplace, but we’re not sure where
    - TV sets, printers, routers, Blu-Ray players, etc.



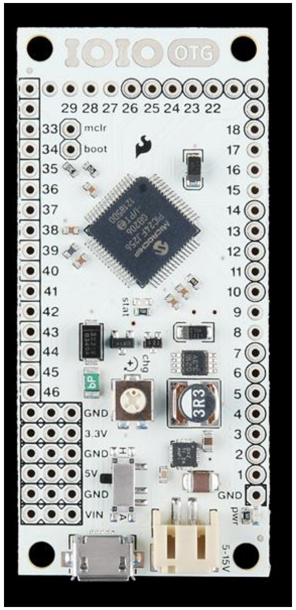
Android is Embedded, not R-T

- So, an out-of-the-box Android device really isn't capable of deadline-based computing
  - It might be fast enough most of the time, but there's no guarantee of service
- We would like to be able to offload the R-T constraints to something else and use Android for the UI
- This is where we come to using an external microcontroller (μC)

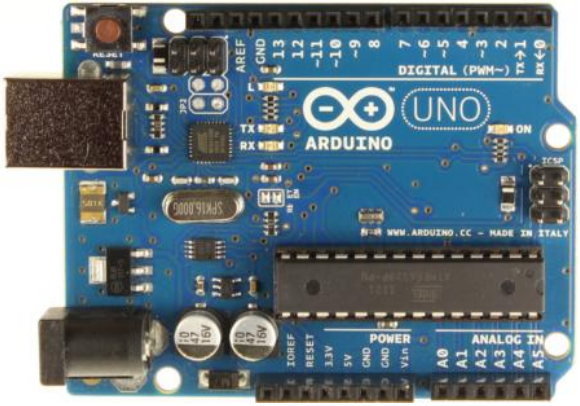
External Microcontrollers

- There are a number of popular microcontrollers these days
  - 8-, 16- and 32-bit variants
- They can't run Linux since they don't have MMUs and lack sufficient RAM
  - "Big" µCs include the 32-bit ARM Cortex M3/M4 with 512K RAM
- They might run an RTOS or they might be bare metal
  - FreeRTOS runs on a number of ARM Cortex M versions
  - Arduino is bare metal
- Examples include:
  - Atmel AVR (Arduino)
  - Microchip PIC24/PIC32 (incl. IOIO board)
  - TI MSP430
  - Various ARM Cortex M0/M3/M4 flavors
- Each of these has its own development environment
  - Tools will typically run under Linux but may require WinDoze or OS/X

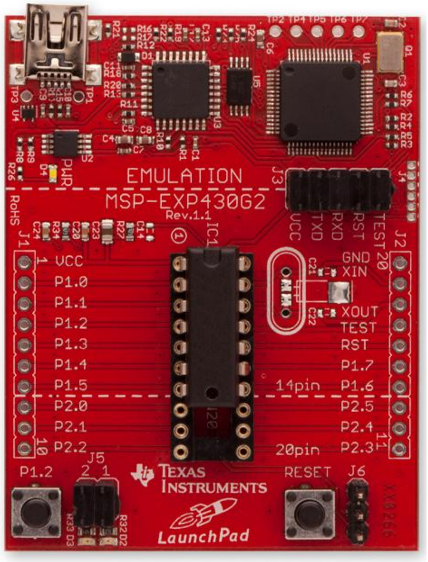
Example Boards



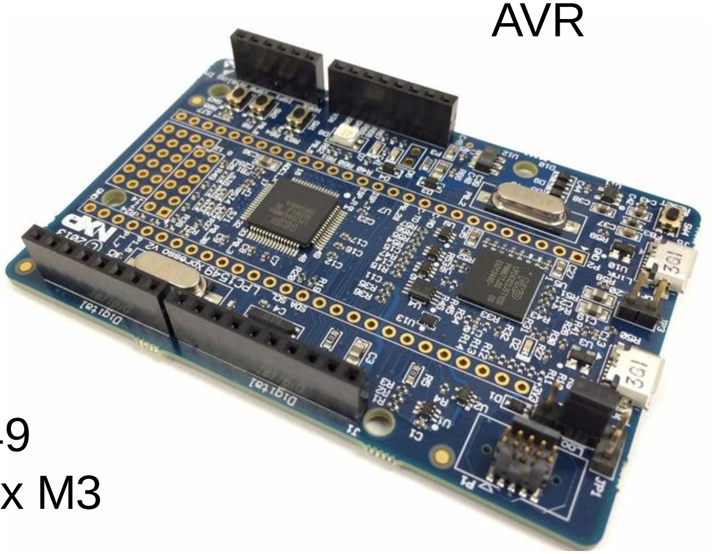
IOIO
PIC24



Arduino UNO
AVR



TI Launchpad
MSP430

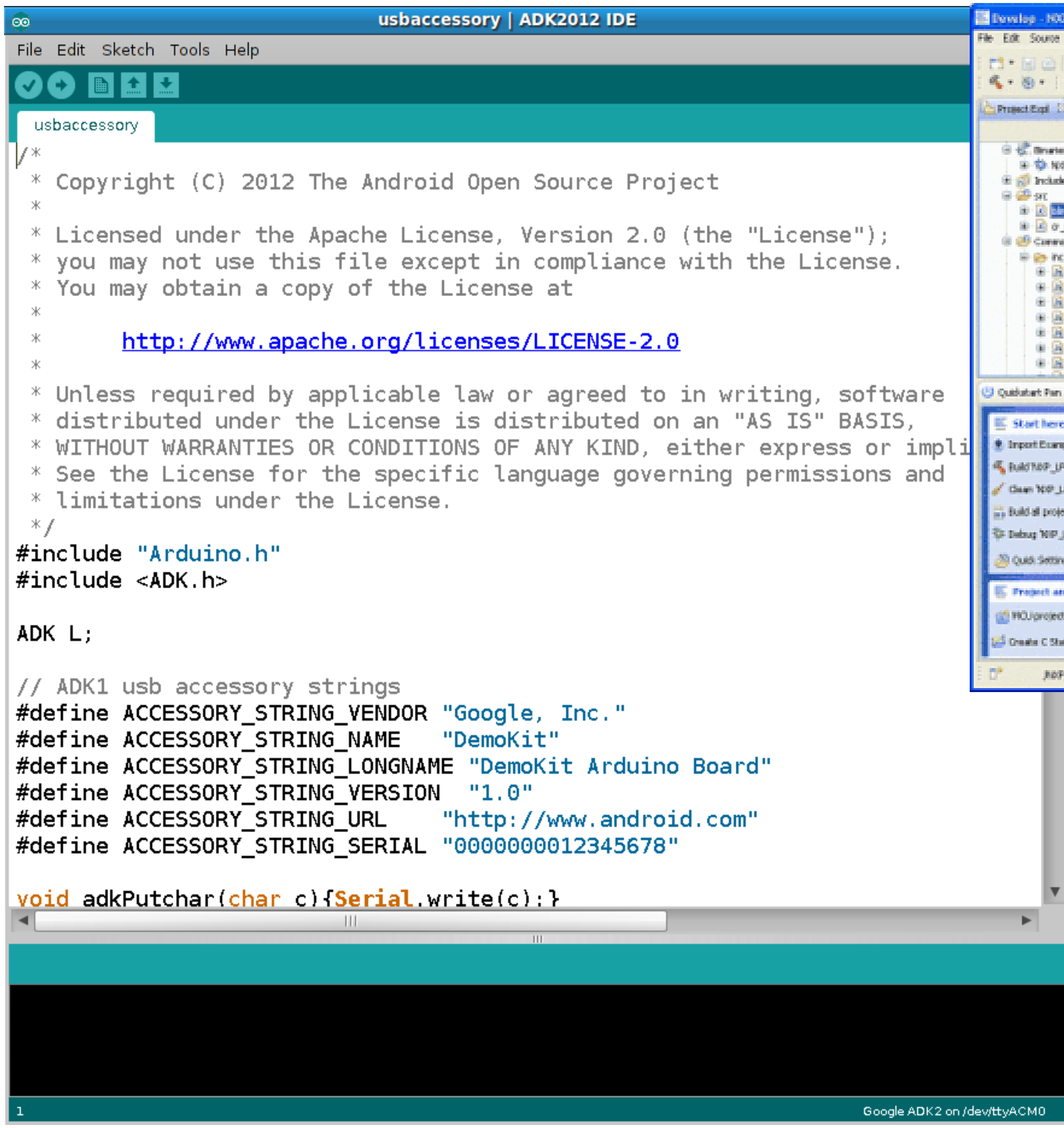


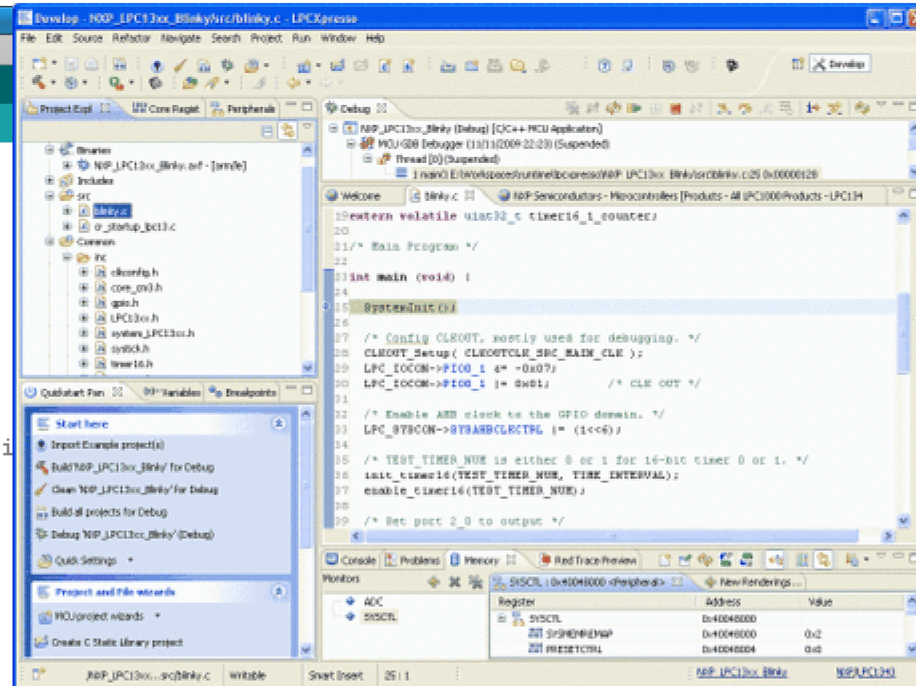
LXP1549
ARM Cortex M3

Two Approaches to the Problem

- There are typically two approaches to using a µC
- We can write code to run on the µC and use the µC to control the data collection and/or control
  - Requires learning the µC IDE and control APIs
    - Some APIs are very simple, others can be almost as involved as the Linux APIs
- Alternatively, we can use a “Firmata” approach
  - We’ll get to this in a moment

Example Development Environments



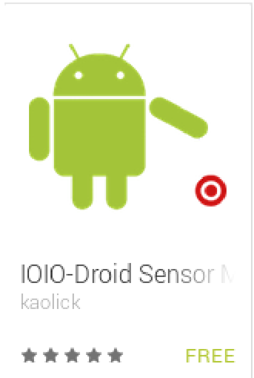


- Many IDEs use standard GNU tool chains
- Some µCs require proprietary tool chains
- Make sure to read the fine print
- Arduino has a development environment for Android

What is “Firmata”?

- Some μCs support a special firmware load similar to the “Firmata” firmware used by the Arduino community
  - Uses a serial interface and simple application to export all of the pins on the μC to the controlling host
- Many examples for the Android side of this on the Play Store
- The IOIO board also uses this approach
  - Unfortunately, not compatible with the Arduino Firmata
- Turns the μC into a dumb peripheral requiring Android to send commands and retrieve data
  - Provides extra I/O to Android, but doesn’t address the time-sensitive control issues

Source: google.com



Source: google.com

Should you Program or use Firmata?

- As with most things in embedded, the answer is "Well, it depends…"
- Using a Firmata approach means you can likely leverage existing .apks from the Play Store
  - But, you force all of the data collection and processing onto the Android device
- Programming the μC takes more time, but allows you to do the time-critical code on the μC and communicate as needed to the Android device
  - You'll likely need to write custom Android code as well to pack and unpack the data
- Software on the μC can operate as polled or interrupt driven or a mix
  - You partition the work as best suits the problem

Connections to the μC

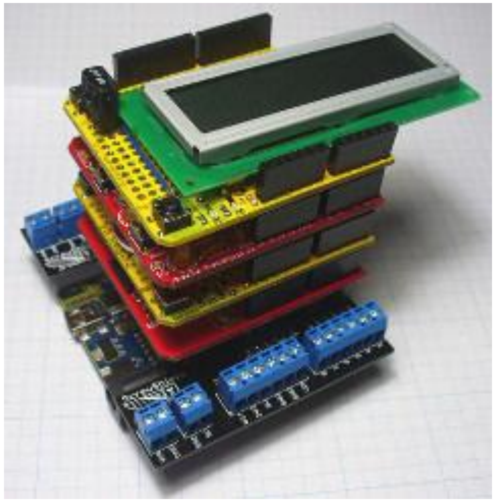
- Many μC boards have a broad selection of connectivity options
  - Serial, Bluetooth, IEEE 802.15.4, USB, Ethernet, Wi-Fi, NFC and more
    - Some of these are native to the μC board and some are via external mezzanine buses
- Regardless of the transport layer, most connectivity boils down to serial communications
  - With the exception of Wi-Fi and Ethernet which look more like BSD sockets

The “Nearly” Universal Connection

- Due to the size and pervasiveness of the Arduino ecosystem, many 3rd party boards have adopted the Arduino pin out
  - Support for I2C, SPI, A/D, D/A, PWM and GPIOs with 3.3V and 5V power and ground
- This gives access to hundreds of plug-in boards known collectively as “shields”

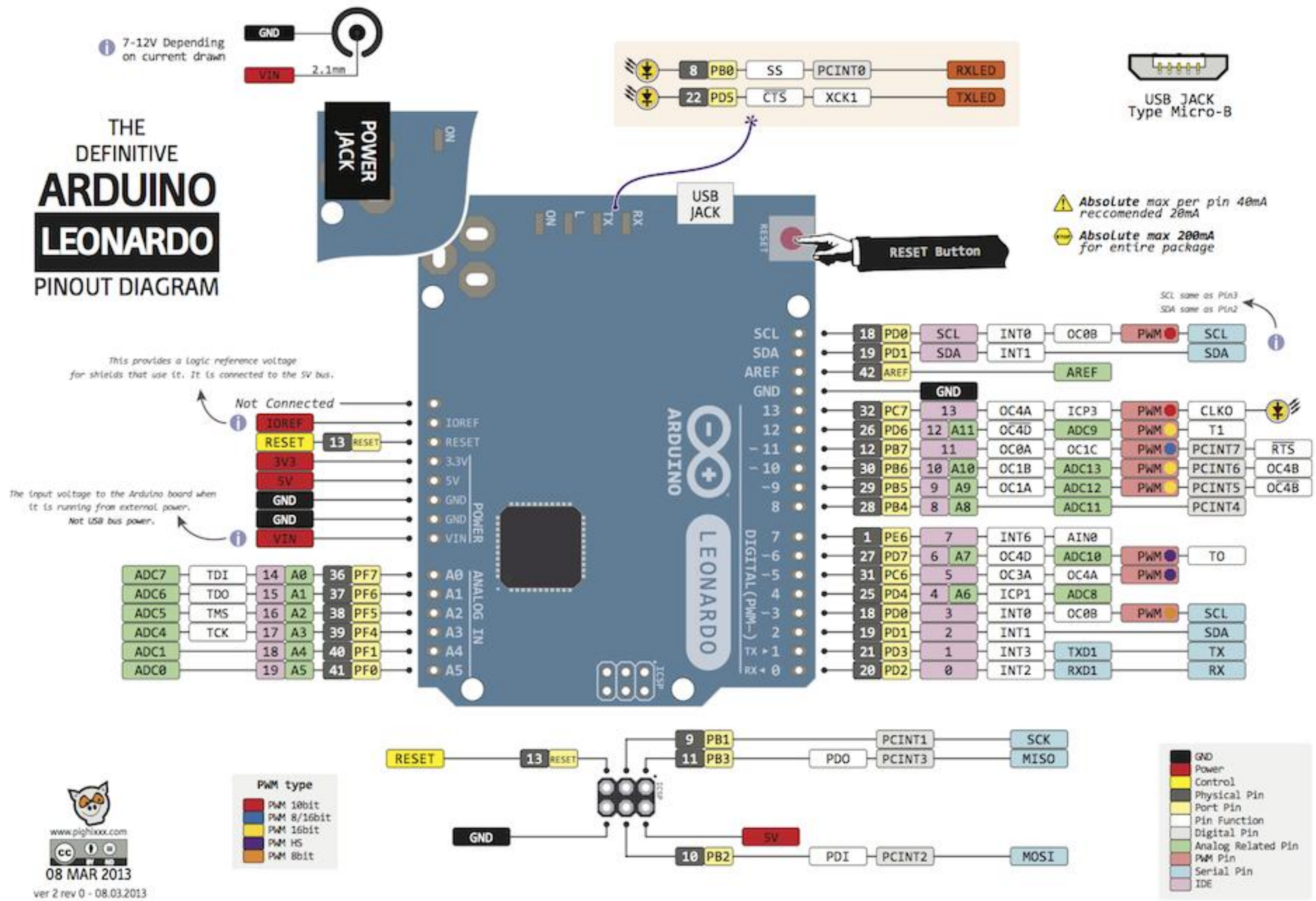
Shields Up!

- A variety of shields are available:
  - Bluetooth, ZigBee, Ethernet, GPS, protoboard, relays, MIDI, SD Card, LCD, motor controllers, and many, many more
- Some shields can be stacked to create complex systems



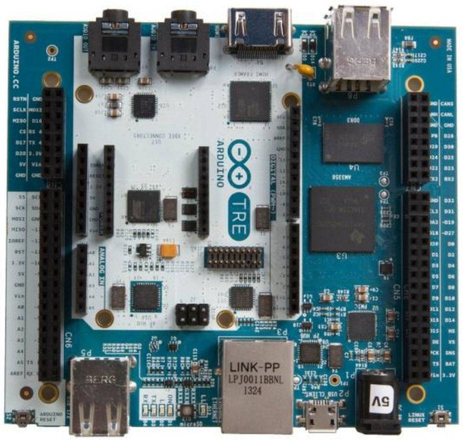
Source: shieldlist.org

Typical Arduino Pin-out

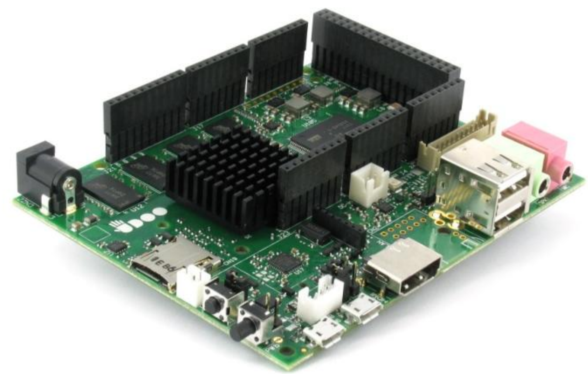


Source: zembedded.com

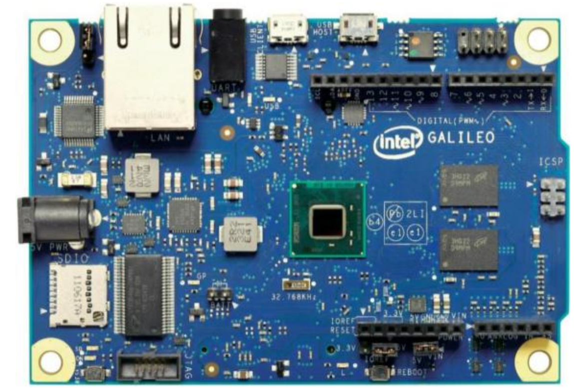
Boards with Arduino Pin-Outs



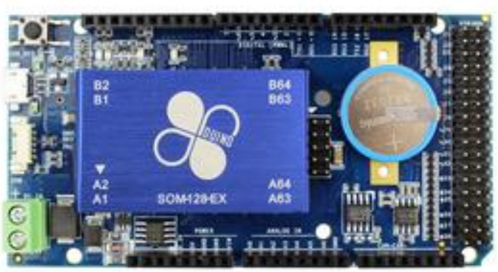
Arduino Tre



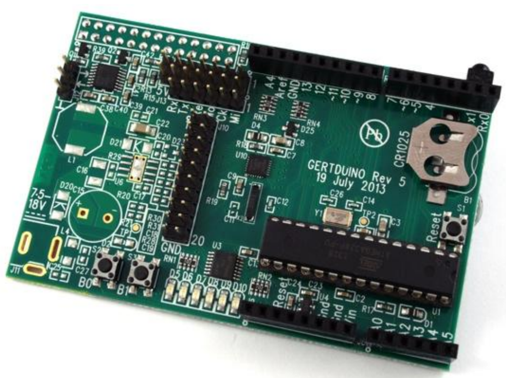
Udoo



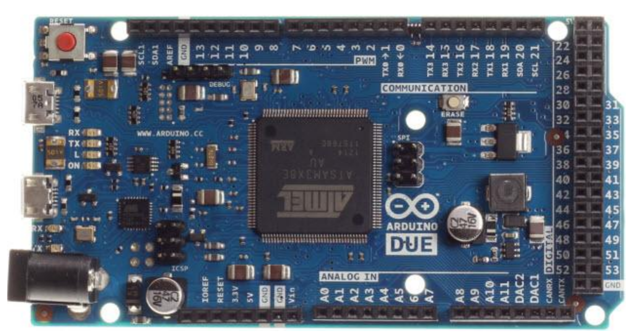
Intel Galileo



86Duino



Gertduino



Arduino Due

Overview of I/O Capabilities

- The major variants:
  - ATmega328 (Uno)
    - 14 DIO (4 with PWM)
    - 6 analog inputs
    - 2 external interrupt lines
    - 1 UART (simple 3 wire)
    - JTAG
    - 2 8-bit, 1 16-bit timer
  - ATmega2560 (Mega2560/ADK)
    - 54 DIO (14 with PWM)
    - 16 analog inputs
    - 6 external interrupt lines
    - 4 UARTS (simple 3 wire)
    - JTAG
    - 2 8-bit, 4 16-bit timers

- Most Arduinos implement a USB to Serial interface for the UART
  - Used to program the Flash as well as for serial I/O
- There is support for Ethernet via the Wiznet 10/100 Mbps W5100 interface (SPI)
  - Wi-Fi and Bluetooth are supported too

Android ADK

- In 2011, Google introduced the Accessory Development Kit (ADK)
  - Used USB to connect Arduinos and IOIO to Android device
  - A standard part of Android since 2.3.4
- In 2012, Google released ADK2 which added Bluetooth **support and support for ARM Cortex M3 (Atmel** SAM3x)
- Really the ADK is just a protocol specification
  - It's been ported to Raspberry Pi
    - Gary Bisson, ABS 2013 -- https://github.com/gibsson

Android and USB

- Android devices still tend to be USB devices rather than USB hosts
- Arduinos w/ USB host shield play the role of USB host and drive the initial connection
- Android detects the addition of a USB device and looks at the handshake to determine the app to run to service the accessory
- USB appears as a serial stream to the accessory
  - You are responsible for packing and unpacking the messages on both sides

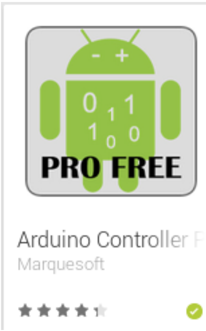
Bluetooth

- Most μCs that support Bluetooth support the SPP
  - ADK2 supports A2DP for stereo audio
- Bluetooth works just like a serial port once the device is paired
  - Bluetooth Smart reduces the issues of pairing with Android devices with Bluetooth Smart support
- There are several apps on Play Store that support Android to μC connection via Bluetooth

Source: google.com

Wi-Fi

- Many µCs support Wi-Fi using the H&D Wireless HDG104 Wi-Fi chipset
  - Hardware TCP/IP core with built-in webserver
    - Data storage via SD Card
- Exports a socket API to the µC
  - Supports both TCP and UDP sockets



Source: google.com

```
void loop() {

  // if there's data available, read a packet
  int packetSize = Udp.parsePacket();
  if (packetSize)
  {
    Serial.print("Received packet of size ");
    Serial.println(packetSize);
    Serial.print("From ");
    IPAddress remoteIp = Udp.remoteIP();
    Serial.print(remoteIp);
    Serial.print(", port ");
    Serial.println(Udp.remotePort());

    // read the packet into packetBufffer
    int len = Udp.read(packetBuffer, 255);
    if (len > 0) packetBuffer[len] = 0;
    Serial.println("Contents:");
    Serial.println(packetBuffer);

    // send a reply, to the IP address and port that sent
    Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());
    Udp.write(ReplyBuffer);
    Udp.endPacket();
  }
}
```

Summary

- Android is a capable platform, but its not easy to natively extend without substantial customization to the hardware, software or both
- Adding external μCs provide additional interfaces not supported by Android and allows us to better partition the problem
  - Without the need to rebuild the AOSP sources
- We shouldn't use Firmata-type interfaces to the μCs unless we have very lax timing requirements
- We have a number of connectivity options so we can chose the connection based on speed and remote access requirements