Embedded Systems:
Concepts and Practices
Part 2

Christopher Alix
Prairie City Computing, Inc.

ECE 420
University of Illinois
April 22, 2019

Outline (Part 2)

- ARM and DSP Architectures
- Software challenges in Embedded Systems
- Key decisions in ES software development
- Low-cost ES Prototyping Platforms
- Trends and opportunities in the ES industry

Embedded System

Definition

- A dedicated computer performing a specific function as a part of a larger system
- High-reliability systems operating in a resource-constrained environment (typically cost, space & power)
- Essential Goal: <u>Turn hardware problems into software problems</u>.

ARM Architecture

Unifying ES Development

32-bit and 32/64-bit variants

Started by Acorn Computers (UK) in 1983

ARM Holdings bought by Softbank in 2016

Core licensees (~500)
(Include ARM CPUs on their chips)

Architectural Licenses (~15)
(Design their own CPUs based
on the ARM instruction set)

ARM Architecture

Unifying ES Development

Licensees include Analog Devices, Apple, AMD, Atmel, Broadcom, Qualcomm, Cypress, Huawei, NXP, Nvidia, Renesas, Samsung, STM, TI, Altera (Intel), Xilinx.

15 billion ARM-based chips sold per year (2016)

Dominant market share
Smartphones (95%)
Computer peripherals (65%)
Hard disks and SSDs (95%)
Automotive (50% overall; 85% infotainment)

ARM Architecture

Unifying ES Development

Brought order to a chaotic industry with dozens of different proprietary processor architectures

Enabled a common set of tools, techniques and technologies to be shared across the ES industry
(one Linux port, one gcc/g++ target, etc., etc.)

8- and 16-bit MCUs still dominant in low-power, low-cost applications, and DSP architectures dominant in signal-processing domains, but ARM basically has everything else.

DSP Architectures

Digital Signal Processing

Special-purpose processors optimized for specific mathematical operations, often in parallel

Able to do specific signal processing tasks substantially faste than a general-purpose CPU, at lower clock speeds (more work for less power!)

The right choice where an application requires lots of repetitive mathematical computations

DSP Architectures

Digital Signal Processing

Texas Instruments

C5500 series: ultra low power, fixed point

C674x series: low power fixed/floating point

C66x series: multi-core fixed/floating point

C66x + ARM: hybrid SoC for complex devices (DSP does signal processing, ARM runs apps)

Analog Devices

SHARC: high-performance floating point

Blackfin: high-performance fixed point

Many variants with different peripherals, etc.

DSP Architectures

Digital Signal Processing

Complicated instruction sets, often VLIW (very long instruction words); numerous functional units with multiple data transfers going on per clock cycle

Example: TI TMS320C674x

- 2 multipliers
- one 32 x 32, two 16 x 16, or four 8 x 8 per cycle
- 6 Arithmetic Logic Units
- Dispatches up to 8 32-bit instructions per cycle

Hardware support for "loop buffers" (allows for highly optimized pipelined loops of short sequences)

DSP Architectures

Software Implications

Code generation is <u>very</u> complicated and best left to a compiler; hand-coding is done very rarely and only in small, extremely time-critical routines.

gcc/g++ support some DSP families, but never as well as the vendor's proprietary compilers--lots of very chip-specific optimizations required.

Best performance achieved by giving the compiler "hints" (loop counts, alignment, etc.)

Algorithm and data structure design is critical.

Software Challenges

"Black Box" Problem

Limited input/output and user interface presents challenges, especially during debugging.

Much embedded software is cross developed—written and debugged in the comfort of a desktop PC, and then downloaded into the system under development for final testing and deployment.

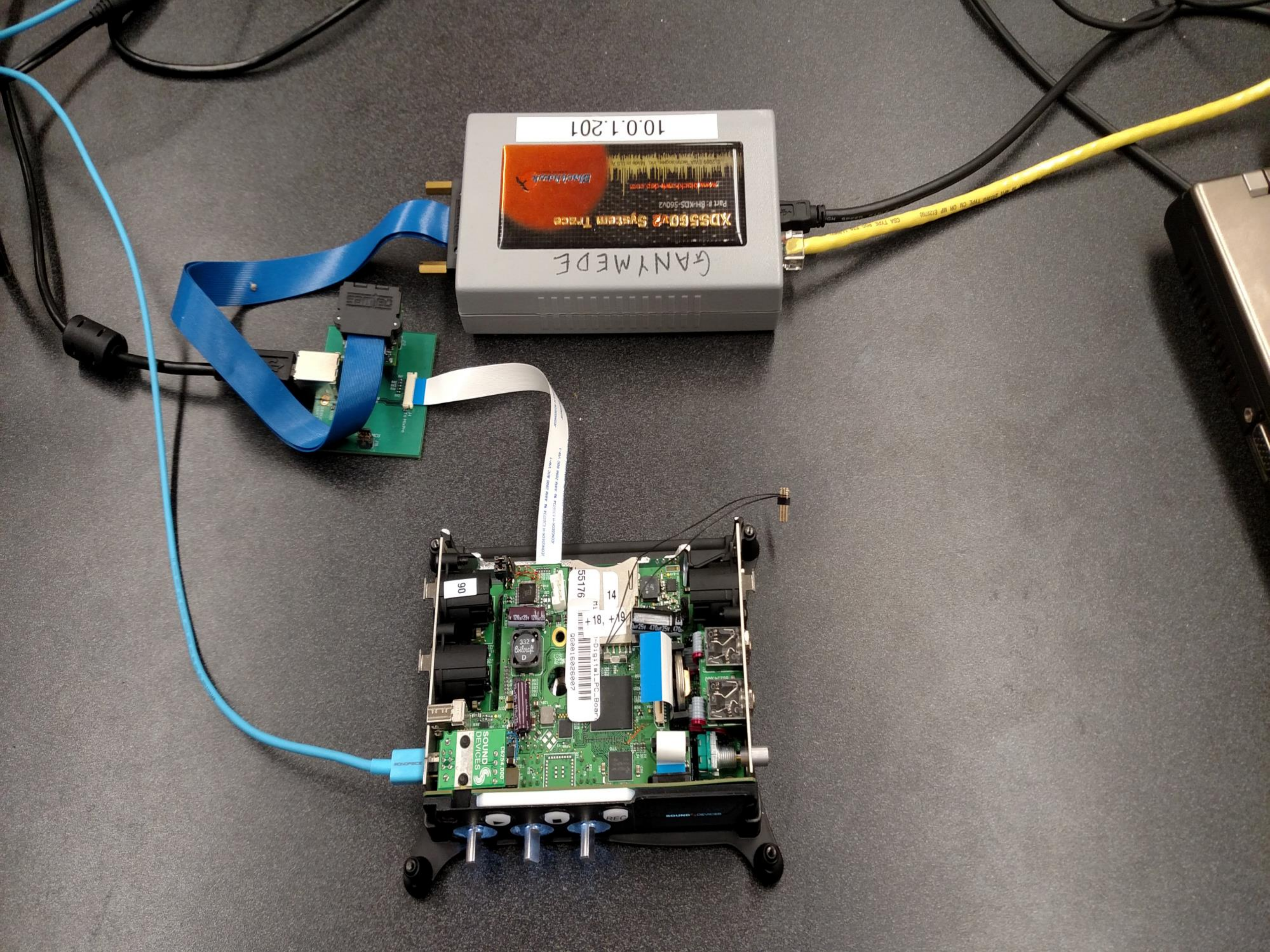
Software Challenges

"Black Box" Problem

Embedded processors typically include a hardware interface (usually JTAG) for loading software and for doing remote debugging from a host computer.

A development version of the hardware is often built first with extra interfaces for testability, which are then stripped out of the final design.

Systems often include a connector for a "debug board" or "breakout board" which includes extra connections for debugging.


10.0.1.201
XDS560v2 System Trace
Part #: BH-XDS-560V2
GANYMEDE
90
55176
14
+18, +19
SOUND DEVICES
REC

Software Architecture

Realtime Requirements

Many ES have tasks that must be performed reliably at a specific rate. (e.g., capture a new audio sample every 21us, open a fuel injector within 10ms of a TDC indication)

Embedded OSes need specialized features to satisfy the need for real-time performance (fine grained priority controls, low-latency interrupt handling, priority inversion, etc.)

Software Architecture

Realtime Requirements

- Real-time performance can generally be achieved with careful software design
- Proving real-time correctness can be hard--”worst” or “corner” cases can be rare, subtle and hard to identify or test
- I/O processors can handle time-critical tasks e.g., Programmable Realtime Units (PRUs) in the TI "Sitara" (AM335x) CPU family

Software Architecture

Realtime Requirements

- Interrupts can help satisfy realtime requirements if used correctly (not locked
out, prioritized sensibly, etc), but…

- Interrupts complicate testing and debugging

- <u>Interrupts don't make the processor faster.</u> A platform that is too slow to keep up with the application requirements in a "main loop" architecture won't be fixed by using interrupts.

Software Architecture

Realtime Requirements

Realtime means “consistent”…

it doesn’t necessarily mean “fast.”

Software Architecture

Data Retention

- Many ES subject to interruption of power without orderly shutdown (battery dies, user yanks the power cord, etc.)
- Filesystems need to be fault-tolerant and able to recover from any state (redundancy, journaling, rewrite-before-erase, etc.)
- Long-lifetime systems run into FLASH and EEPROM write-cycle limitations (use load-leveling, RAM disks, etc.)

Software Architecture

”Revenge of the Kilobyte”

- In the desktop world, computing power has grown faster than software complexity. For all but the most compute intensive tasks, performance limitations are rarely a factor.

- Even in the server world, throwing more processors at a problem is usually cheaper than extensive software optimization efforts (hardware is cheaper than programmers!)

Software Architecture

”Revenge of the Kilobyte”

- “High Performance” desktops...
  1980: 16KB RAM, 4 MHz clock, 1 core
  10MB HD, 1200 bps modem
  1990: 16MB RAM, 25 MHz clock, 1 core
  1GB HD, 10 megabit Ethernet
  2019: 32GB RAM, 4 GHz clock, 6 cores
  4TB SSD, Gigabit Ethernet

- Improvement by orders of magnitude (10^{3+} speed, 10^5 storage, 10^{6+} memory)

Software Architecture

"Revenge of the Kilobyte"

- In the ES world, cost and power constraints require "making every cycle count."

  Intel 8-Core i9-9900KF CPU
  Power consumption: 95 W

  1 AA battery = 2.5 W-H
  Solar (PV) panel (2018) = 320 W/m^2

  Vitatron E10A1 (...?)
  2.5 W-H (10.4 years), 27uW

Software Architecture

”Revenge of the Kilobyte”

- Developing for many embedded systems is like developing for a 1980-era desktop...
- But at least we've got faster machines to run the development tools on!

Software Architecture

”Revenge of the Kilobyte”

Doing more with less...

Memory/Processing Trade-offs
(look-up tables v. calculations)

Profiling and Optimization

Native Code (C/C++, Assembler)

"Stupid Math Tricks"
fixed-point
shift-add multiplication
reciprocal division
alternatives to complex functions

Software Architecture

OS or not?

Simple “bare metal” systems run a single main-loop program that does everything. Usually compiled C/C++, with some assembly language in the simplest, most cost-sensitive systems.

In between, there are (typically proprietary) quasi-OSes designed specifically for embedded applications, typically on specific processors
(e.g., TI-RTOS, OS-9, LynxOS, FreeRTOS)

More complex systems use an OS, often Linux.

Software Architecture
GNU/Linux

Free/Open Source Software (FOSS) technologies (GNU compiler tools and GNU/Linux operating system) are ubiquitous in embedded system development

"Free as in freedom" access to source code simplifies debugging, minimizes development risks and enables future enhancements

"Free as in beer" lack of licensing fees provides additional pricing flexibility/profit margin, critical for very low cost devices!

Software Architecture
GNU/Linux

Lots of complexity and overhead, but can be trimmed down with custom kernel configuration and a "Linux from scratch" approach to system building. (5-second boot times achievable for some systems)

ARM port unified and reworked since 2014—great improvement despite some transition hassles. "Device tree" model replaced a lot of kernel configuration, reduced boot time and cleaned up code. Fewer special cases, more consistency.

Mature, comprehensive real-time support in the mainline kernel

Software Architecture

GNU/Linux

Upsides:

Support for many processors and peripherals (often no need to do custom device driver development)

Powerful "distribution builder" tools like Yocto/OpenEmbedded and Buildroot make it easy to build an entire Linux system

Lots of leverage; essentially the same workflow and tools as desktop/server Linux development

Software Architecture

GNU/Linux

Downsides:

Fairly high minimum hardware requirements limit it to "real" processors (ARM, x86); limited support for DSP families; not an option for lower-end uCs

Constant updates require either a commitment to continuous integration, or significant "catch-up" work to migrate to new kernel releases

Boot time can be improved, but still unacceptable for "instant-on" applications

Software Architecture

GNU/Linux

Risks:

Combinations of versions, platforms, distributions, processors, system architectures are nearly infinite--unlike in the mainstream software development world, you may be the only person in the world trying to do what you're trying to do.

Maximize your leverage by starting from "known good" reference designs. There's safety in numbers...try to run with the herd rather than reinventing the wheel! Use your innovation time on the features of your product, rather than trying to get the bootloader to work...

Software Architecture
Android

Operating system for mobile devices developed by Google and the Open Handset Alliance (mostly for mobile phones, but is applicable to many non-phone ES projects too!), 2008

May, 2017: announced reaching 2 billion monthly active users worldwide

GNU/Linux plus Android-specific tools

Applications written in Java and run on Google’s proprietary ART virtual machine

Software Architecture

Android

Android phones and tablets from many vendors; reference designs; OpenEmbedded support

Advantage: maturity, commercial acceptance, broad hardware support. libhybris: leverage Android binary device drivers under Linux

bootlin.com (was free-electrons.com)

Stand-up training, w/~2000 slides online under a Creative Commons license. Android, Linux system development, Linux device drivers, etc.

Software Architecture
RTOSes

Lightweight, fast, efficient systems, usually specific to a specific processor or processor family
- TI-RTOS (was SYS/BIOS, was DSP/BIOS)
- FreeRTOS (Acquired by AWS in late 2017)
- Analog Devices VDK
- VxWorks, QNX, OS-9, LynxOS

Provides support for process scheduling, interrupt handling, memory management, interprocess (and interprocessor) communication, etc.

Often combined with a PSP (Platform Support Package) which provides rudimentary device drivers--sometimes more trouble than they're worth but at least useful as sample/starter code

Thoughts on Software

- Typically 10:1 (or higher) ratio of software engineers to hardware engineers on many ES projects
- Selection of development and debugging tools, in concert with hardware debugging support
- Software is key to debugging the hardware, and vice versa--groups must work closely together at times
- User interface design is important to the quality and usability of the resulting product

Thoughts on Software

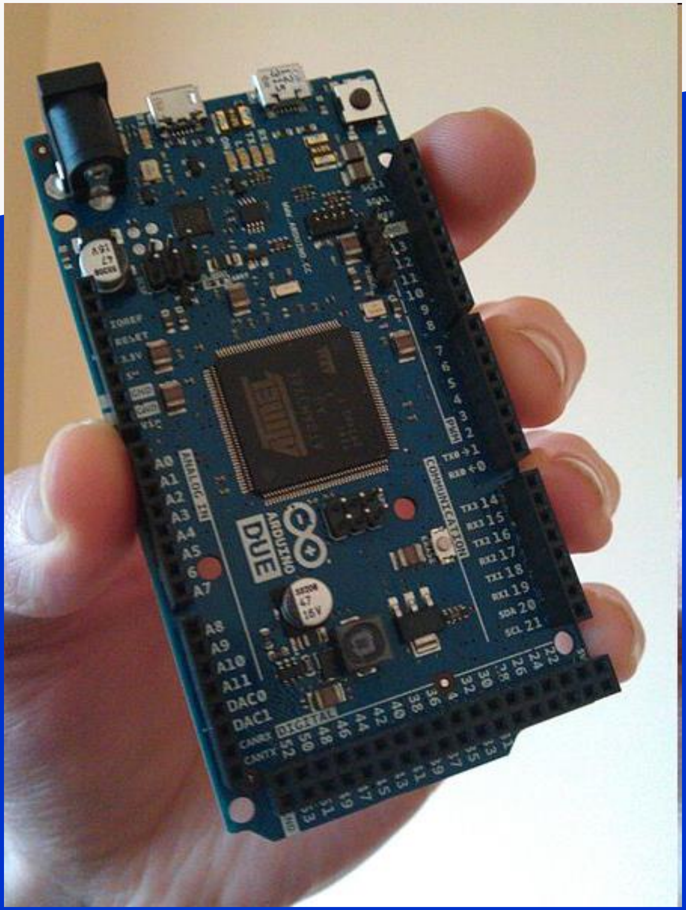
Software makes the hardware work (or not work)

Open-source tools common in ES world

Debugging environments vary widely based on the capabilities of the hardware

Software is easier to change than hardware, but quality is equally important

Arduino



Atmel MCUs up to ARMs

$15-$50

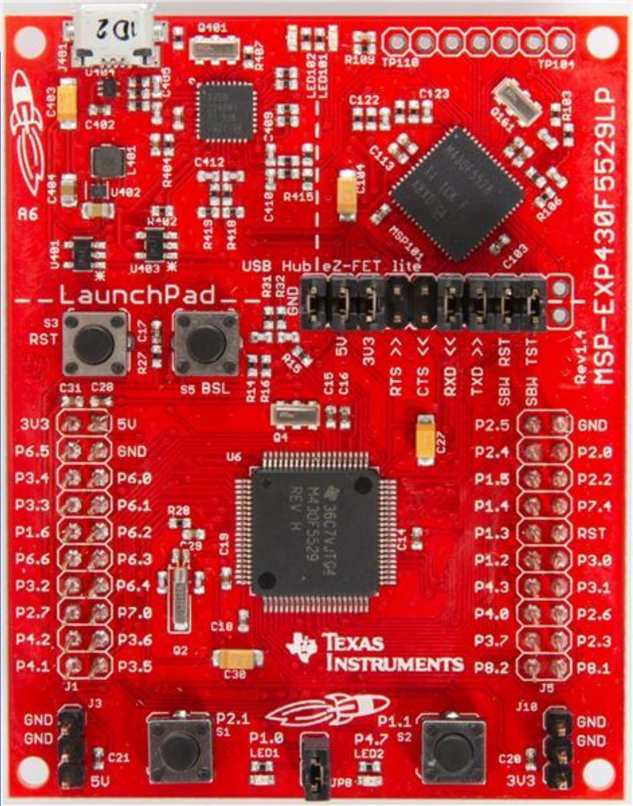
Wide variety of “shields” for expandability

Open-source hardware design

> 1M sold

www.arduino.cc

TI MSP430 LaunchPad
$12



(One of Many)

MSP430F5529 16-bit MCU

JTAG, UART, USB

gcc or TI compilers (free)
support MSP430 compilation

"BoosterPacks" for expandability

Design files available

Target: small battery-operated applications

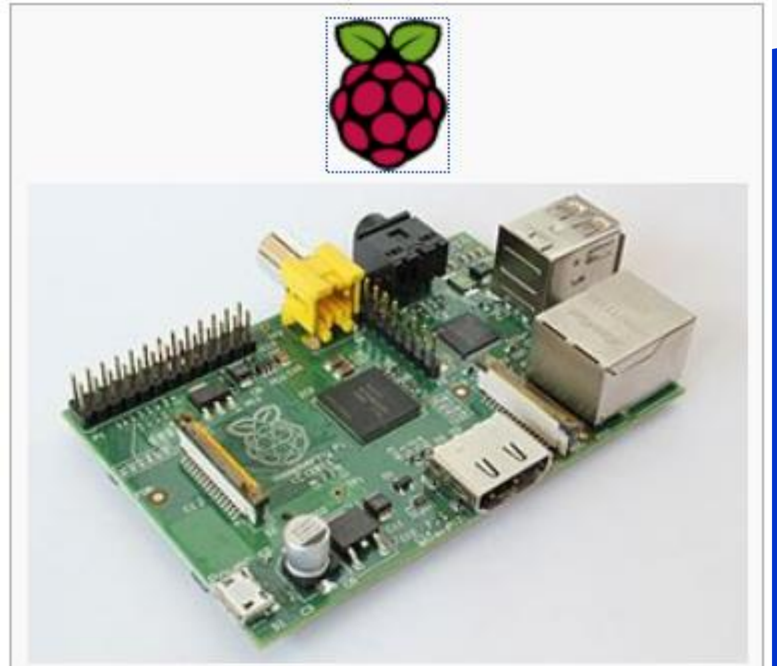
TMS320C6748 LCDK
$195

Up to 456MHz 32-bit DSP
VGA Video Output
Audio Codec
USB x 2

Also available for OMAP-L138, which is an ARM core and DSP core on the same chip

Schematics Available (Reference Design)
Target: multimedia processing

Raspberry Pi B
$35



"Raspberry Pi" Computer Model-B Rev1

700 MHZ ARM CPU

512 MB RAM

10/100 Ethernet

HDMI Video Output

2 USB 2.0 Ports

MPEG-2 and MPEG-4 Video Support

Third-party peripheral modules available

Broadcom processor; some peripherals proprietary and poorly documented

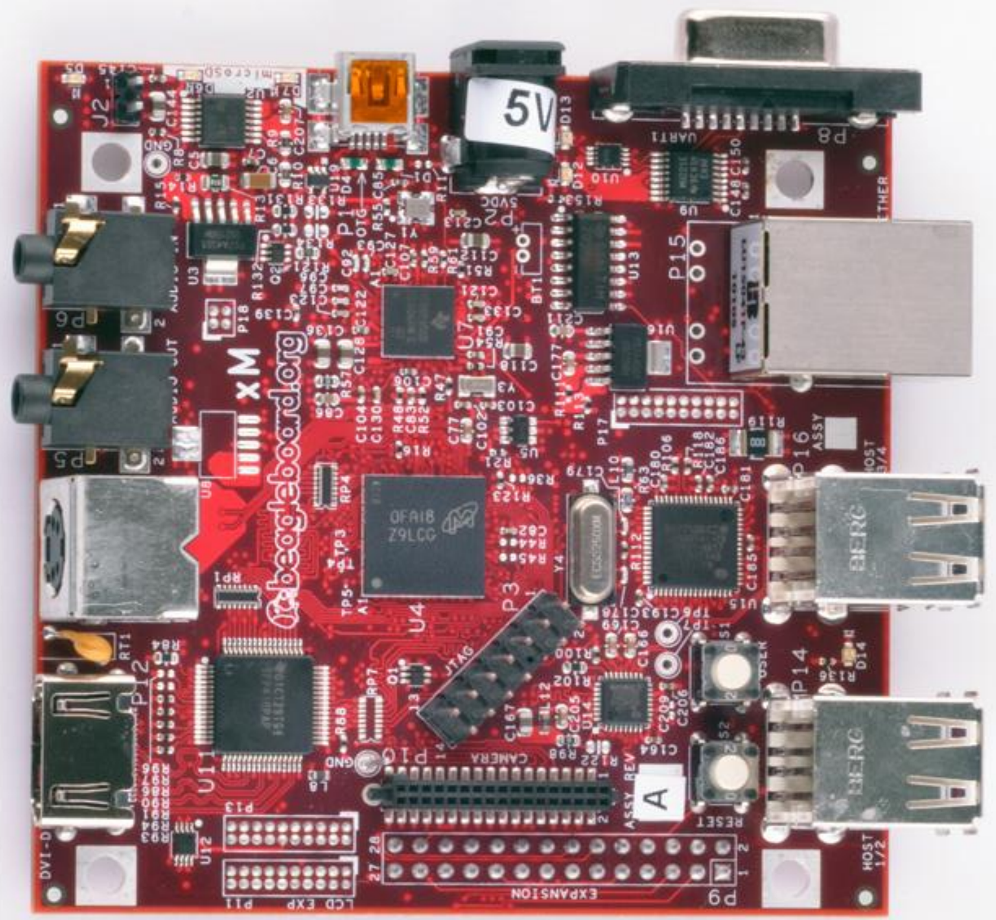
~10M sold as of 3Q 2016 (many to schools)
www.raspberrypi.org

Beagleboard-xM

$150

TI DM3730
P-O-P Memory
SD Card Slot
S-Video and HDMI
Audio Line In/Out
USB, Ethernet

Published open source hardware design



Gumstix Overo

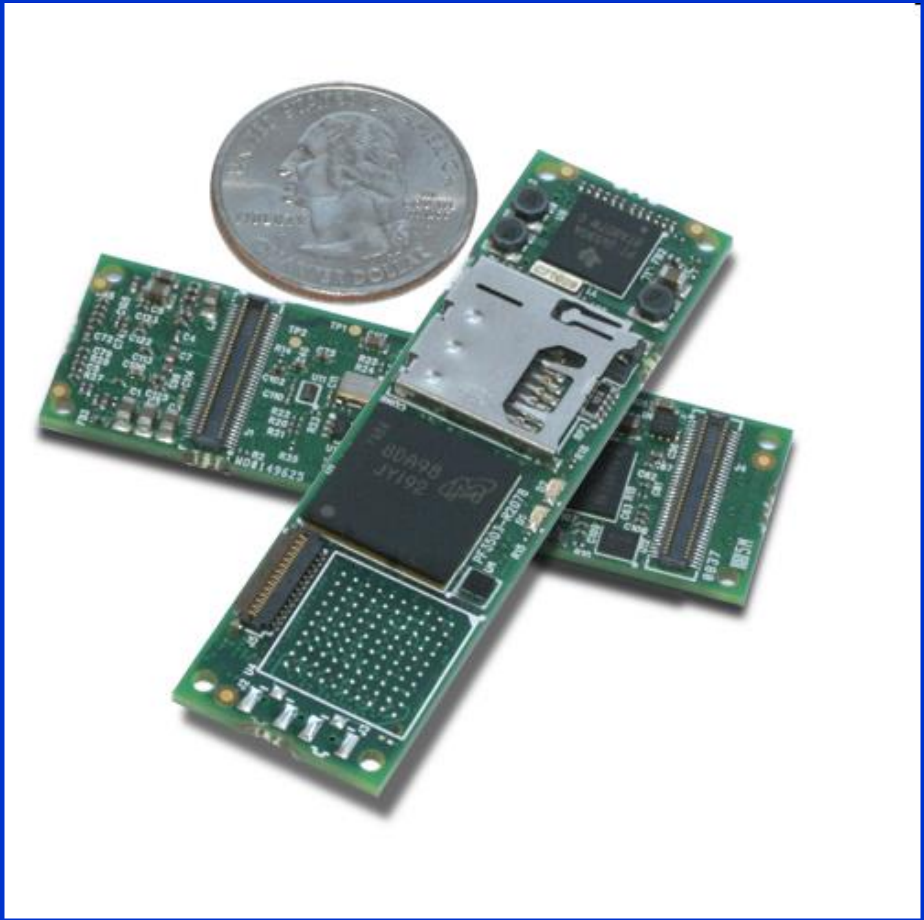
$120-$220

TI DM3730
P-O-P Memory
Micro-SD Card Slot

Optional Wi-Fi

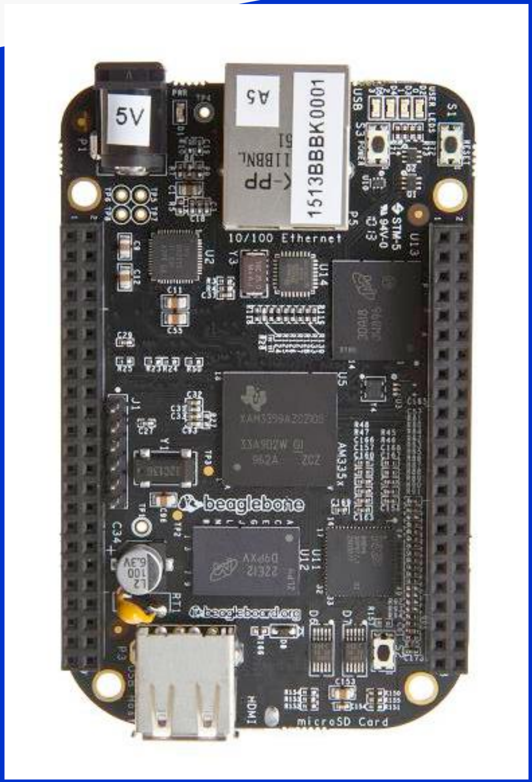
Single +3.3V Supply

All I/O on two high density connectors



Beaglebone Black

$45



1GHZ TI AM3359 "Sitara"
ARM Cortex-A8 CPU w/2 PRUs
Ethernet, USB x 2
512MB RAM, 4 GB FLASH
Boots Linux from SD card

Wide variety of expansion “capes”
Full schematics and design files available
500,000+ sold as of late 2012

www.arduino.cc

Wandboard Quad+

$140

1GHZ NXP i.MX6 CPU
Quad ARM Cortex-A9 Cores
GB Ethernet, WiFi, BT, USB x 2
2GB RAM
3 graphics engines
Boots Linux from SD card

Realtime video processing, image recognition,
High-performance graphics applications

Wandboard PICO-PI

$80

1GHz NXP i.MX7 CPU
ARM Cortex-A7+ Core
Gig Ethernet, Wi-Fi, Bluetooth
4GB Flash, 512MB RAM

Designed specifically for Google's "Android Things"
(IoT development ecosystem based on Android)

Target applications are robotics, remote sensing,
UAVs, home automation systems, etc.

ES Development

Software Skills

Linux (desktop skills translate to ES work!)

C/C++ (Linux Kernel is still straight C, but
full C++ is usually the right choice
unless code space is at a premium!)

gcc, g++, gdb, git
Understand the toolchain end-to-end
(even assembly listings and load maps!)

Yocto/OpenEmbedded and Buildroot

Eclipse (4th-gen open-architecture I.D.E.)
    CCS (TI DSPs), XSDK (Xilinx), more

ES Development

Software Skills

-- Familiarity with common CPU architectures (ARM, MCU and/or DSP families)

-- Bootloaders and the boot process
(Getting from power-on to "Hello World")
U-Boot, Barebox

-- For mobile development: Android, iOS

Closing Comments
Observations

Managers and engineers on ES projects need a solid understanding of hardware and software issues

Hardware and software development are done in parallel, by multiple groups, so agreement on standards and protocols and a clear specification is critical to keep the project moving forward

Right the first time: An extra hour of design time can save days or weeks of development time.

Closing Comments

Observations

Once exclusive to HPC, "Artificial Intelligence" applications (really just fast search and signal processing) are becoming a big driver of innovation and performance requirements in the embedded space. Voice-input and computer vision for autonomous vehicles (drones and road vehicles) are hot areas.

"Embedded" now includes mobile (2B units a year), consumer, wearable, control systems, sensing, autonomous, IoT…

What part of ECE/CS isn't embedded at this point?

We're going to need a better name!

Closing Comments

Opportunities

Easier than ever for small companies to bring sophisticated embedded systems to market

Hobbyist-class development platforms
FPGA-based designs
Outsourced PCB fab and assembly
F/OSS (including FPGA cores…opencores.org)
3D printing, CNC laser cutting and machining
Kickstarter, e-Commerce, global communities

Closing Comments

Opportunities

Unlike most engineering students, ECE/CS students don't need to play with "toys."

The tools ECEs use in the classroom, the living room or the maker space are the tools being used in the "real world."

Make! Hack! Create! Get hands-on experience!

Then, never stop learning...

Questions

Thank you!

alix@ieee.org