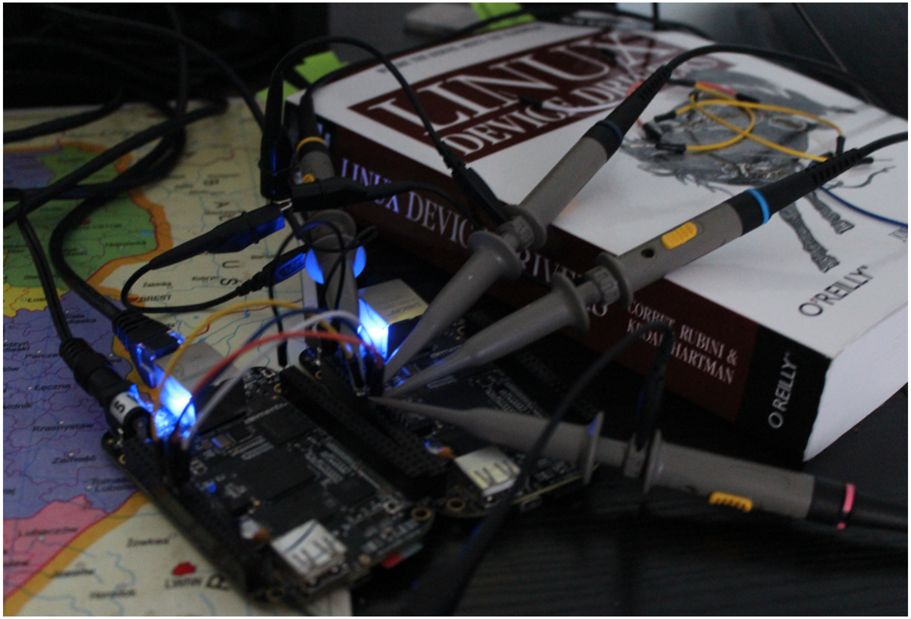
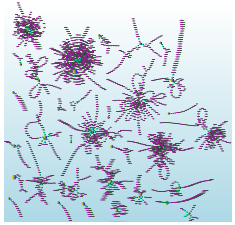
IoT Without a Net: A Practical Guide To Working With Microcontrollers The Open Source Way



Presentation Outline

Who Am I?



Gentoo Linux Developer

OpenEmbedded contributor

BeagleBoard GSoC Mentor

Grumpy toolchain guy (part-time)

Geeky software tool maintainer

http://bbb.io/gsoc
https://github.com/sarnold
https://github.com/VCTLabs

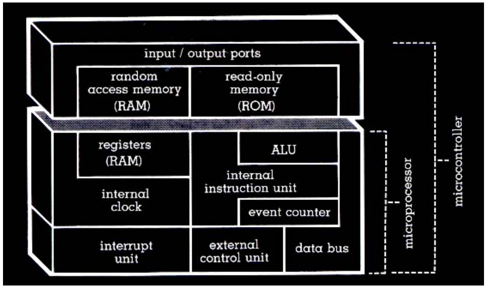
Systems Architecture guy

Principal Scientist VCT Labs

Exactly What Kind of Hardware Are We Talking About?

- Primary Characteristics
  - Stand-alone or separate real-time CPU core(s)
  - Can also come in combination or hybrid configurations
  - Requires firmware flashing to do something
  - Includes both hard and soft processor cores
  - Interfaces for debug, RAM/peripherals, Linux runtime
- Typical Applications
  - IoT, industrial, automotive, consumer, hobbyist
    - Fixed and mobile sensor platforms
      - Wearables, remote instrumentation, mesh networks
      - Autopilots (drones, rovers, UAVs) and robotics
      - Vehicle navigation, entertainment, communication
    - Machine control (3D printing, milling, process control)
    - Door/entry control, locks, power, LEDs, appliances

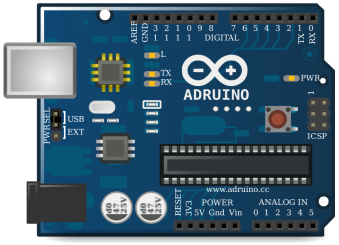
Example Reference Architecture



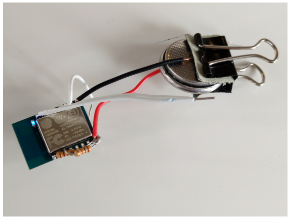
"A microcontroller (or MCU, short for microcontroller unit) is a small computer or System on Chip (SoC) in a single integrated circuit containing a processor core, memory, and programmable I/O peripherals (may also include program memory)."

[1] https://en.wikipedia.org/wiki/Microcontroller

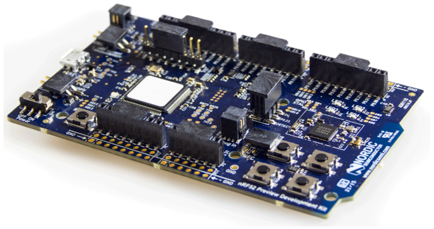
Example Boards



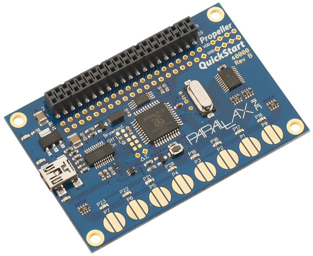
AVR/ARM: Arduinos



Xtensa LX-106: ESP8266



Cortex-M4: nRF52 DK



P8X32A microcontroller

Microcontroller Architecture / CPU Families

- 4 most "common" architecture families in DIY and FOSS
  - 8051, PIC and AVR are Harvard architecture, which uses separate memory spaces for RAM and programs, while ARM is von Neumann architecture (program and RAM share the same memory space)
  - ARM is a 16 or 32 bit architecture, others are byte (8-bit) architecture
  - 8051 and PIC have limited stack space - limited to 128 bytes for the 8051, 8 words or less for PIC
  - 8051, AVR and ARM can directly address all available RAM, while PIC can only directly address 256 bytes
  - 8051 and PIC need multiple clock cycles per instruction, while AVR and ARM can execute most instructions in a single clock cycle
  - AVR and ARM have great open source compilers, libs, examples
- Still around: PowerPC, MIPS, STM, TI, Toshiba, Freescale/NXP, etc
- Combination/Hybrid and Special Purpose Architectures/Implementations
  - PRU-ICSS / PRUSSv2 - Programmable Real-Time Unit and Industrial Communication Subsystem
  - DSP - Digital Signal Processor
  - FPGA - Field Programmable Gate Array
  - Massively Parallel - Parallax Propeller, Parallela, Transputer

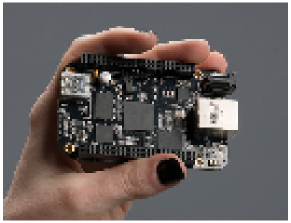
Hybrid and "Combo" Boards

Most embedded Linux boards have co-processors for graphics or network acceleration but here we're talking about something a little different. While the bulk of microcontrollers are stand-alone, there are several options available with both an ARM host and separate microcontroller cores.

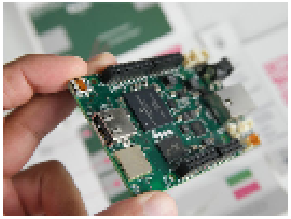
- General purpose microcontrollers (not DSPs)
- Can be Cortex (ARM), AVR, FPGA, other

Examples:

- TI Beagleboard / Beaglebone boards (AM335X ARMv7 w/ PRU cores)
- Altera "hybrid" Arm/FPGA boards (socfpga ARMv7 w/ 1 CycloneV FPGA)
- Udoo / Udoo Neo boards (i.MX6 ARMv7 w/ 1 ARM Cortex-M core)


BeagleBone Black

DE-0 Nano SoC


Udoo Neo Full

What About Software Tools?

Just like every (beagle) dog has his/her day, every arch has its toolchain...

nerdboy in #gentoo-embedded

The typical embedded Linux cross-toolchain is built to target processor architectures different from the build host and has 4 main parts:

- a cross compiler for C and C++ (in the case of gcc, possibly more languages)
- a cross binutils (the build tools needed by the compiler)
- a set of libraries and headers for the supported languages
- a set of kernel headers for the target platform

The (cross) compiler prefix (triplet) contains several bits of important info:

- machine-vendor-OS (where OS can be two fields and vendor can be left out)

```
# gcc-config -l
[1] armv7a-hardfloat-linux-gnueabi-5.4.0 *
[2] armv7m-hardfloat-eabi-5.4.0 *
[4] mips64-unknown-linux-gnueabi-5.3.0 *
[8] x86_64-pc-linux-gnu-5.4.0 *
```

Toolchains, SDKs, and Architectures

Bare-Metal and Firmware

Each micro-controller/real-time core has an architecture and instruction set, which any toolchain must support (or at least a viable subset). Most have both vendor support and a gcc port, although in some cases the gcc port might be new and essentially still under development (eg, the TI PRUSS toolchain).

The toolchain itself is mainly the compiler/linker/loader and a standard C library (of which there are several). In order to build anything "fancy" you also need a Software Development Kit (or SDK). Sometimes you can live with either the compiler runtime and/or libc, but a good set of libraries with documented interfaces is always a Good Thing. Depending on the target board there is typically a vendor SDK and (hopefully) several open source alternatives.

Places to start:

- AVR-Ada Blog by Tero Koskinen
    - tkoskine bitbucket repos - Tools for AVR and ARM in multiple languages
- Embedded Project Blog - by Stefan Wendler
    - wendlers github installer - Best installer for propeller tools

You should always do your homework (for a real project) or just pick one and try it and see...

Vendor vs. Open Source Tools

Vendor boards typically have a vendor SDK and vendor toolchain options (sometimes including GCC) whereas open source hardware generally relies on a GCC toolchain (with some interesting exceptions). The more "mainstream" the architecture, the more likely it is to have a mature GCC option.

"Common" architectures

- ARM: Cortex-M/R "official" toolchain is GNU ARM
- AVR: Atmel Studio is based on GNU AVR

"Oddball" architectures

- TI PRU-ICSS: "official" toolchain w/ full debug support is CGT and CCS
- Parallax Propeller (original) toolchain is based on SPIN

The rub: FPGAs

- FPGAs tend to be more vendor-specific (both hardware and toolchains)
- Open source firmware toolchains do exist (some components may be missing and/or immature)
- Each board typically has its own interface to host side and may require custom Linux integration, etc
- Peripherals implemented on the FPGA side may also need custom DT and driver support for Linux, etc

Examples: nRF52 DK, ESP8266, FPGA, PRU-ICSS

Great, we have a toolchain/SDK and a devel-kit/eval board - now what?

- Compile some firmware/bootloader or try the vendor flash demo
- Connect any necessary cables, etc (for nRF52 connect the NFC antenna)
- Hook up a serial/debug interface (microUSB or serial pins or even host USB)
- If needed, apply power (eg, flip the ON/OFF switch)

Each SoC/board should have "standard" interfaces, along with any vendor-specific things for flashing firmware. The nRF52 DK uses a mass-storage interface for this, while Altera SoCs have a "USB Blaster" host port.

- Flashing your firmware is board/vendor-specific
  - nRF52 supports mass-storage/JLink/UART
  - ESP8266 supports USB/serial (esptool.py)
  - DE-0/DE-1 supports u-boot, USB Blaster, and JTAG (power-on only)
  - BeagleBone loads PRU code on boot (from /lib/firmware)
- Debug from x86 (desktop, laptop) or ARM (RPi3, BB, Udoo)
  - JLink connected to USB, debug console (possibly BLE)
  - Provide 5v/3.3v power as needed

Examples: Altera socfpga

DE-0 Nano and DE-1 SoC Kits (CycloneV FPGA plus ARM Cortex-A9)

Another "rub": the Altera socfpga boards are a different species, since the hardware peripherals are literally split, ie some are connected to the ARM HPS side, and others are connected to the FPGA side. On the HPS (Linux) side all you get is ethernet and a serial console with an FPGA firmware blob. What?!?

Linux - FPGA Integration

The big issue is the FPGA demos aren't buildable without rolling back to (very) old toolchains/u-boot/kernel (or a lot of fiddling and poking). Why? There's an important piece of integration glue missing/broken from the vendor tools and the u-boot side is virtually undocumented (see here for the missing README content that was lost in patch limbo).

Finally, there's the problem of too much documentation (really!). As several university courses use socfpga boards to teach processor design, such as ECE 5760 at Cornell, they make a good starting point.

Best advice to get started is the LinuxOnArm wiki and DE1_SOC_Linux_FB Demo project. Use the vct-socfpga-bsp-platform and 4.4-altera kernel to build a rootfs with the right kernel and u-boot patches (follow the DE1 FB readme for details).

Examples: PRU-ICSS, TI AM335x, AM437x, AM571x

Hmm, Kernels and kernels and PRUSS interfaces...

nerdboy in #beagle-gsoc

Given how long beagleboards and beaglebones have been around, there have been several long-lived TI and BeagleBoard.Org kernel branches, and you may still run across projects, demos, or examples that still require a 3.8 or 3.14 kernel. The PRU interfaces have evolved over that time, so using the latest available versions of PRU sample projects and the TI-staging kernels should "match".

- Old kernels/PRU firmware use the old UIO PRUSS interface
- Newer kernels/pru-software-support-package use the Remoteproc interface
- Old TI kernel branches include 3.8 and 3.14, newer branches are 4.4 and 4.9
    - New beaglebone patches should appear in the bb-kernel mainline build
- To get started, read Using the C language to program the am335x PRU
- Clone the pru-software-support-package repo, install the CCS compiler package, and either use a bb-kernel build with debian rootfs or build everything with the vct-beagleboard-bsp-platform
- Follow along (as much as possible) with the TI Training in Hands-on Labs

Examples: PRU-ICSS Cont.

GCC vs. TI PRU C compiler issues and resources

- PRU ELF file format - merged, both PRU-GCC and TI toolchain use 0x90 for machine code
- TI ELF PRU Relocations - *Work is ongoing to make Binutils and TI ELF relocations compatible*
- GCC PRU ABI and TI PRU C ABI are slightly different

See also:

- TI PRU-ICSS wiki "portal" page / resources
- PRU-ICSS Getting Started Guide

Interesting Applications:

- BeagleBone - beaglelogic, PRUDAQ, SPI slave driver (title pic)
- Altera socfpga (not the demos): DE1-SoC-Sound, I2S audio using opencores
- ESP8266: NodeMCU and microPython

How To Choose?

It depends. What do you want to do? Educate yourself? Make a cool project? Prototype a product idea? Automate something?

- Education - maker space, LUG group, home
  - Very little open source/popular hardware is truly mainlined
  - Pick a board off the LinuxOnArm wiki
  - Pick a board used at the hacker space or recommended by a friend
  - Pick a board with a large community and lots of projects
- Commercial product - kickstarter, customer, boss
  - Understand the project requirements
  - Evaluate some eval boards
  - Understand the hardware limitations
  - Evaluate the kernel and runtime needs
  - Evaluate performance/other reqs vs. BoM costs

Where to Go Next

nRF52 DK and toolchain, flash wrapper

- https://www.nordicsemi.com/eng/Products/Bluetooth-low-enery/nRF52-DK
- https://developer.arm.com/open-source/gnu-toolchain/gnu-rm
- https://github.com/VCTLabs/nrfjprog-wrapper

BeagleBone/BeagleBoard info, PRU toolchain, Yocto BSP manifest

- https://eewiki.net/display/linuxonarm/BeagleBone+Black
- http://elinux.org/Ti_AM33XX_PRUSSv2
- https://github.com/VCTLabs/vct-beagleboard-bsp-platform

DE-0 Nano SoC info, updated demo projects, BSP manifest, vendor refs

- https://eewiki.net/display/linuxonarm/DE0-Nano-SoC+Kit
- https://github.com/VCTLabs/DE1_SOC_Linux_FB
- https://github.com/VCTLabs/DE1-SoC-Sound
- https://github.com/VCTLabs/vct-socfpga-bsp-platform
- https://github.com/altera-opensource/linux-refdesigns

License and Thanks!

Author:	Stephen L Arnold
FOSS Hat:	Gentoo Linux Developer
Contact:	nerdboy@gentoo.org
Revision:	0.0.1
Date:	2017-02-27, 19:53 PST8PDT
License:	CC-Attribution-ShareAlike
Copyright:	2017 Stephen Arnold
Other:	All other trademarks and copyrights belong to their respective owners.