Building an IoT-Class Device

Igor Stoppa

Embedded Linux Conference / Open IoT Summit Europe
October 2016

V 0.1.0

Disclaimers

Opinions expressed here represent the author’s view.

All Logos and Trademarks represented belong to their respective owners.

Summary

- What is an IoT device?
  Why would I want to build one?
- Defining the playground:
  purpose, environment, use cases, threats.
- Identifying the constraints: time, materials, means.
- System design: HW, SW, development environment
- The gory details:
  - HW / SW selection
  - Identifying and integrating the key components
- Consideration about optional features.

What is an IoT device? Why would I build one?

IoT device:
(semi) autonomous agent, producing and/or consuming information, mostly by interacting with other IoT agents.

Not a new concept:
home, industry automation have existed for a long time.

What is new: plummeting costs, pervasive connectivity.

An unprecedented opportunity for data collection and automation.

IoT devices are not born equal

Topology - Nodes vs. Leaves:

- Multi-purpose vs. single task
- Beefier vs. leaner specs
- Some Linux flavor vs. RTOS or minimalistic Linux

Security threats:

- Direct exposure to the Internet vs. tamer intranet
- Open services running
- Risk of physical tampering

Goals and Constraints

Goals

- Rapid development
- Avoid investing time in distro technology that is not close to the use-case.

Constraints

- Certain use-cases are not yet standardized.
- Supporting them means choosing among competing options.

The unit we want to build

- **High-End Leaf**
  has functionality of its own, others can contact it
- **No extreme cost optimization**
  Wherever possible, relies on stock components
- **Ease of maintenance**
  Keeping it up-to-date is simple and it is not labour-intensive
- **Self-contained**
  Supports development for its own applications

Choosing HW and Distro

- **Prefer a familiar Distro, even if suboptimal:** it will pay off when debugging or looking for help.
- **Prefer boards supported by the Distro directly:** A HW vendor will eventually stop upgrading custom images.
- **Prefer HW fully supported by mainline kernel:** it has a larger user base than the single Distro you have chosen.
- **Adjust security measures to the worst case scenario, in case of a breach. Data sensitivity?**

Why this approach?

Minimize the Total Cost of Ownership

The TCO includes:

- Bill of Materials
- Training on new tools (unless you really want to)
- Development time (see previous point)
- Maintenance of custom code (it *will* bitrot)
- Tracking of upstream bugs, especially security

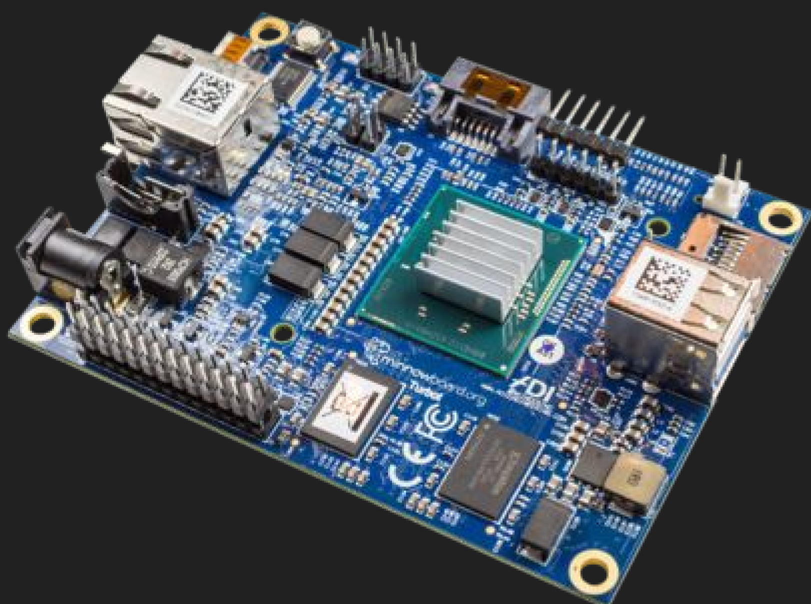
A PC-like device can exploit a wealth of resources, unavailable on any alternative solution.

When does this make sense?

- Small team, limited bandwidth - **focus on the use case**.
- Time constrained development. **Reuse, do not reinvent**.
- Little expertise in certain areas (ex: distro maintenance)
- HW cost not critical (**small batch / high profitability**).
- **Need to be future proof.**
- User not a threat to security (DRM-free, no local threats).
- Desire to tap into large base of developers familiar with mainstream distros, rather than requiring specific training.
- Don’t want to do unnecessary rebuilds of unrelated components.

Practical Example

MinnowBoard Turbot [1]



- PC-like UEFI BIOS
- Small form factor
- Passive cooling
- Low power
- Boots regular distros (Debian, Ubuntu, RH)
- USB3:
  easy to expand

Alternatives: RaspBerry PI 2 & 3

Distro selection

Distro chosen: Debian

But it could have been anything sufficiently familiar: Ubuntu/Snappy, OpenSuse, RedHat, etc.

Do not spend time re-learning how to get the base distro up and running unless it brings major gains.

Distro configuration - 1

- Dump the standard installer image onto an USB stick
- Proceed with the installation on an SD card (MinnowBoard doesn’t have internal storage), following the typical installation procedure for the chosen distro.
- Use the most minimalistic configuration available: those components that are specifically needed for the required use case will be installed in a 2nd phase

Distro configuration - 2

- Install, configure and enable ssh services.
  Follow best practices to ensure that the service is installed both properly and securely. Ex: [2].
- Configure and enable the firewall.
  Allow only what is needed. Block the rest.
  Even use a script generator is better than skipping this step. Ex: [3]

Make a copy of the SD card: it will be useful as base for either creating other projects, or as restore point.

Designing the Application

Make the service available through ad-hoc interface.

Anything that fulfills the need will do:

data stream, archive with records, http page, …

Profile and perform optimizations only in the face of a problem that can be clearly identified and measured.

Sensors/Actuators integration

Embedded vs Discrete

- Many choices of bus: I2C / SPI / Camera / …
- High bandwidth, when needed.
- Minimal space occupation.
- Same electrical circuit as the SoC.
- Comparatively low power - good for battery life.
- Special debugging rig.
- Direct electrical interface

- One bus type: USB
- USB3 can have high throughput.
- Can take up significant space.
- USB hub can provide electrical decoupling.
- High power scenario, typical with USB hub.
- Debug on a normal PC.
- USB hub as protection.

Practical Example - continued

Create an IP Camera

for monitoring purposes.

Choosing the sensor:

- simple USB camera available through v4l2.

Streaming Solution:

- Gstreamer server through secure connection.

Practical Example - configure

- Install gstreamer-tools on the sd card.
- Set Up a gstreamer source using the selected USB camera sensor [10].
- Establish a permanent ssh connection between the IP camera and its consumer [6].
  Not efficient, but easy to setup and verify.

Avoid growing the attack surface.

- On the consumer side, establish the gstreamer sink that will use the data produced on the camera [11].

Practical Example - Pain Points

- Not fully standardized.
- Multiple Competing Solutions.
- Risk to invest resources into a losing solution.

<u>Is there a real need that justifies the feature(s)?</u>

- Advanced Software updater
- Extra security features
- Interoperability APIs

Practical Example - SW Updater

- Distro package manager:
  - Very well tested
  - Can cause conflicts, with partially completed updates
- Image based approach:
  - Reliable
  - Bandwidth Intensive
- Advanced updaters:
    - Container-based: ostree/flatpack [7] - snappy[8]
    - Diff-based: ClearLinux [9]
  - Very optimized
  - Fairly new, with limited use-base

Practical Example - Extra Security

Examples: IMA, SMACK, SELinux, Apparmour

IFF used correctly, they can greatly harden the device

Is there enough competence to use them proficiently?

Most likely they will (greatly) hinder the development.

Does the worst case scenario justify their use?

Practical Example - Interoperability

Various options:

- OCF [4],
- Node-RED [10]
- Soletta [11]

The cost: creating and maintaining bindings

What is the additional value?

Be opportunistic

Questions?

Thank you!

Backup Info

References

[1]. http://wiki.minnowboard.org/MinnowBoard_Turbot
[2]. http://www.cyberciti.biz/tips/linux-unix-bsd-openssh-server-best-practices.html
[3]. http://iptables.xn--rzeniczak-sbc.pl/generator_setup.php
[4]. https://openconnectivity.org/
[5]. http://www.einarsundgren.se/gstreamer-basic-real-time-streaming-tutorial/
[6]. https://www.everythingcli.org/ssh-tunnelling-for-fun-and-profit-autossh/
[7]. https://ostree.readthedocs.io/en/latest/ - http://flatpak.org/#page-top
[8]. https://developer.ubuntu.com/en/snappy/
[9]. https://clearlinux.org/features/software-update
[10]. http://nodered.org/
[11]. https://solettaproject.org/

Example - commands

[10]. **Starting the gstreamer capture pipeline:**

```
gst-launch-1.0 v4l2src device="/dev/video0" ! videoconvert !
videoscale ! video/x-raw,width=800,height=600 ! avenc_mpeg4 !
rtpmp4vpay config-interval=3 ! udpsink host=127.0.0.1 port=5200
```

[11]. **Showing the gstreamer video feed:**

```
gst-launch-1.0 -v udpsrc port=5200 caps = "application/x-rtp\,\
media\=\(string\)video\,\ clock-rate\=\(int\)90000\,\
encoding-name\=\(string\)MP4V-ES\,\ profile-level-id\=\(string\)1\,\
config\=\(string\)000001b001000001b58913000001000000012000c48d8800cd
3204709443000001b24c61766335362e312e30\,\ payload\=\(int\)96\,\
ssrc\=\(uint\)2873740600\,\ timestamp-offset\=\(uint\)391825150\,\
seqnum-offset\=\(uint\)2980" ! rtpmp4vdepay ! avdec_mpeg4 !
autovideosink
```