Edge Computing with RISC-V Platforms Running XIP Linux

Vitaly Wool / Maria Wool, Konsulko AB
About us: Maria

- Photographer, video and QA engineer
- Started off with Android CTS testing
- Currently living in Malmö, Sweden
About us

- Has been working with embedded Linux since 2003
- Currently living in Malmö, Sweden
- Staff Engineer at Konsulko Group
- Managing Director at Konsulko AB
About this presentation

- RISC–V overview
- XIP overview and Linux XIP
- RISC–V and Edge computing
- XIP on RISC–V: what's done and what's to be done
- Edge computing with XIP on RISC–V
RISC–V
RISC-V: what it is and what it is not

- open source hardware Instruction Set Architecture (ISA)
  - RISC (reduced instruction set)
  - Royalty free for any chip manufacturer
  - Developed by UC Berkeley
    - V stands for 5-th RISC design from Berkeley
    - Standard maintained by non-profit RISC-V foundation

- It’s not a CPU implementation nor a company
## RISC-V vs ARM

<table>
<thead>
<tr>
<th>Feature</th>
<th>RISC-V</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction set</td>
<td>RISC (load/store)</td>
<td>RISC (load/store)</td>
</tr>
<tr>
<td>32 bit supported</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>64 bit supported</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>128 bit supported</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Licensing</td>
<td>free and open source ISA</td>
<td>proprietary ISA</td>
</tr>
<tr>
<td>Community</td>
<td>emerging</td>
<td>well established</td>
</tr>
<tr>
<td>XIP for 32-bit MMU</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>XIP for 64-bit MMU</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>XIP for 32-bit !MMU</td>
<td>in progress</td>
<td>yes</td>
</tr>
<tr>
<td>XIP for 64-bit !MMU</td>
<td>ready, not merged</td>
<td>no</td>
</tr>
</tbody>
</table>
XIP in a nutshell
XIP: execute in place

- The code is executed directly from persistent storage
  - as opposed to executing from RAM
  - everything should be resolved at compile time

- Media
  - Typically NOR flash
  - QSPI

- More common for RTOSes
  - compile time resolution is not a limitation
  - no userspace/kernelspace, monolithic application to be flashed
XIP on Linux

- Kernel XIP is supported in Linux
  - Support for ARM32 goes way back
  - RISC-V support has been merged in 5.13

- Userspace can be run as XIP as well
  - requires special filesystem
XIP advantages

- Less RAM needed
  - Usually up to 10x smaller RAM footprint
  - Sometimes no RAM at all is needed

- Lower idle power consumption
  - May be crucial for IoT running on battery

- Shorter boot time
  - No copy on boot

- Faster execution
  - QSPI flash
XIP obstacles

- You can’t write to flash and execute from it at the same time
- However, you can write to flash using special tricks
  - Code copied/executed from RAM
  - No other code may be executed during that time
- XIP requires more space on flash storage
  - At least kernel code can not be compressed
- All addresses are defined at compile time
  - Which may be a security compromise
RISC–V and Edge Computing
Edge computing

- Extends the traditional cloud based IoT model
  - client data is processed at the periphery of the network
    - literally “closer to the edge”
- reduces volumes of data to be moved
- allows for a better utilization of many “small computational powers”
- enables AI in IoT where the connection is weak / intermittent
Edge computing with RISC-V

- RISC-V is becoming increasingly popular in modern IoT designs
  - relatively high computational power for MCUs
  - open source hardware design, no royalties

- RISC-V support in many RTOSes is still somewhat lacking

- Edge computing applications are relatively complicated
  - harder to debug in traditional single-app RTOS environment
Many RISC-V MCUs are supported by Linux kernel
  - notably, the K210 family

Cheap development boards available
  - e.g. Maixduino
RISC-V MCUs and IoT

- It's tempting to run Linux on RISC-V MCUs for IoT
  - faster development
  - easier debugging
  - Shorter time to market

- However, such designs are relatively expensive
  - Linux requires a lot of RAM to run
  - Linux is generally more power hungry than RTOSes

- Some way for painless cost optimization has to be found
Edge computing with XIP
XIP is the answer

- Reiterate the main issues with RISC-V/Linux edge computing
  - cost of the design
  - shorter lifetime on battery

- XIP allows for drastic RAM reduction
  - also to reduce footprint

- XIP allows for (almost) zero power consumption in idle

- XIP technology is transparent for application development
  - i.e. you can prototype without XIP enabled
Real life example

- Application: Image capture / recognition

- Hardware: Maixduino [link]
  - RISC-V Dual Core 64bit, with FPU
  - 8MB SRAM (of which 2 MB for AI)
  - 8 MiB NOR flash

- Software: Linux
  - kernel 5.15
  - buildroot–based userspace (newlib)
Linux (no XIP)

- Kernel
  - Flash consumed: 400k
  - RAM consumed: 1170k + dynamic

- Root filesystem
  - Flash consumed: 500k
  - RAM consumed: 1Mb–4Mb

- Application
  - Flash consumed: 560k
  - RAM consumed: <2M
Linux (XIP)

- **Kernel**
  - Flash consumed: 1240k
  - RAM consumed: 252k + dynamic

- **Root filesystem**
  - Flash consumed: 1M
  - RAM consumed: < 1M

- **Application**
  - Flash consumed: 560k
  - RAM consumed: <2M
Conclusions

- XIP goes very well with what RISC-V designs have to offer
- XIP technology allows to reduce design costs and device power consumption for RISC-V IoT designs
- XIP technology makes it possible to run Linux in low-cost Edge computing solutions
Thanks for your attention!

Vitaly.Wool@konsulko.com