softprise
CONSULTING OÜ

Linux on MCUs:
from marginal to mainstream?

Vitaly Wool
Softprise Consulting OÜ 2015

Microcontrollers

- Key features
  - Tight integration of components
  - Low power consumption
  - Low price
  - **Very** limited RAM and persistent storage
- Appliances
  - Automation
  - Digital Signal Processing (DSP)
  - “Internet of things”
  - Automotive
-

Microcontrollers and Linux?

PRO	CONTRA
Free software and tools	Bigger footprint
POSIX-compliant	Longer boot-up times
Portable and extensible	Stronger requirements on hardware
Many top-notch developers	No/few commercial distributions
Very strong community	The Linux community is skeptical about MCUs

Work accomplished

- 2.6 based EmCraft distribution for MCUs
  - Works on many MCUs
  - Presumes external (D)RAM
- Softprise tweaks to run on DRAM-less system
  - Kernel XIP
  - Userspace XIP
  - Compress data sections even in XIP kernel
    - Done separately, not as a part of the build
- Only for STM32F27/STM32F29

Summary

- Linux on an MCU is possible
  - But the MCU should be powerful enough
- No mainline support
- Some 2.6 based vendor kernels exist
  - e. g. from EmCraft
- Relatively easy with external (D)RAM
  - EmCraft Linux distribution works out of the box
- Possible without external (D)RAM
  - With large enough SRAM
  - And with large enough tweaking

Moving forward

- Community acceptance targeted
  - Forward port required
- More configurability
  - Support for other microcontrollers
  - Compile out redundant/unnecessary parts
- Optimize heaviest remaining things
- Streamline XIP support
  - .data section compression as a part of *xipImage* build
  - Select best compression algorithm

Community acceptance as a target

PRO	CONTRA
Leveraging community support	Possible know-how exposure
Maintenance cost reduction	Will have to play by the rules
New features become available	Possibly bigger footprint
new/better compiler and tools	Large one-time effort

Catching up with the community

- Objectives
  - linux-tiny git as a base for forward porting
  - Emcraft's implementation to port
- Obstacles/additional effort
  - Use standard clock framework
  - Creating proper defconfig
- Results
  - Forward port mostly complete
  - The code will be made available soon
  - 840k .text, 132k .rodata, 86k .data (BT, no TCP/IP)
  - Further optimization highly desirable

Configurability and redundancies

- *printk()* format strings take a lot of space
  - Implement dictionary?
- Much of kernel *lib/* code is not used
- Better *inline* handling
- ProcFS code is bloated
  - Add compile-time option to only select what's needed
- Memory management asks to be simplified
  - Too complex page allocation for CONFIG_SLOB
- Kernel IP stack is way too heavyweight
  - Too hard to optimize, try picoTCP instead

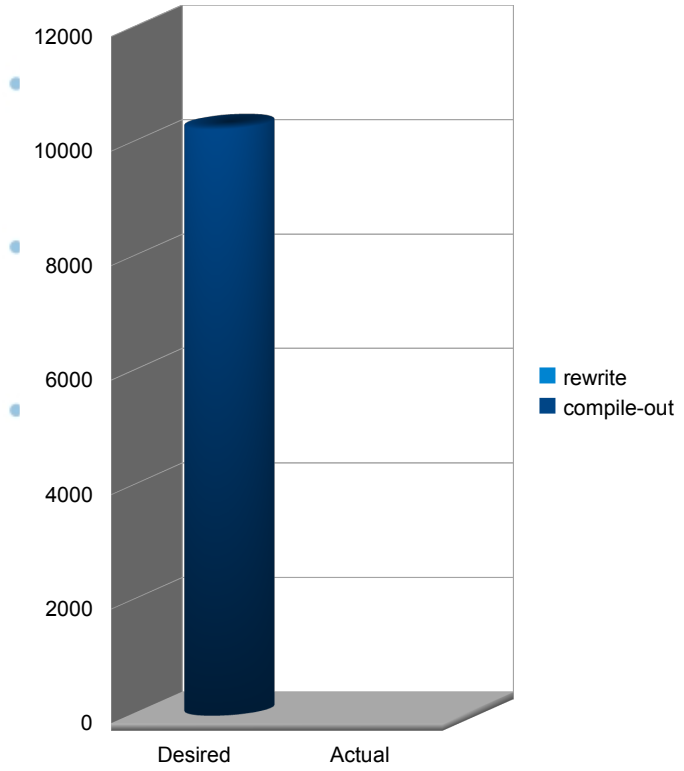
Code bloat hunt

- Try to optimize biggies first
- Run simple scripts to figure out which ones are the largest
  - (*for f in `cat object-file-list`; do echo Analyzing $f; arm-none-eabi-objdump -h $f | grep "\b0 .text"*)
- Analyze what can be done to go down in size
  - Configure/compile out completely
  - Partially compile out
  - Partially rewrite
    - e. g. get rid of heavy #define's and inlines
- Decide on whether it's worthwhile and move on

Printk string dictionary

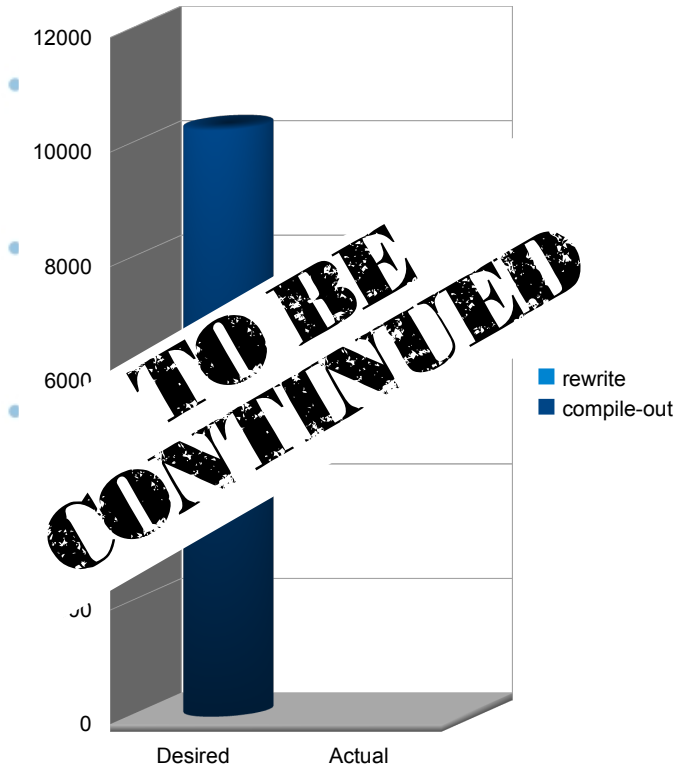
- Tempting target:
- Ideas
  - Compile out printks below certain priority level
  - “dictionarize” format strings
- Dictionary size: 3k
- Still under way

TO BE DONE



Streamlining ProcFS

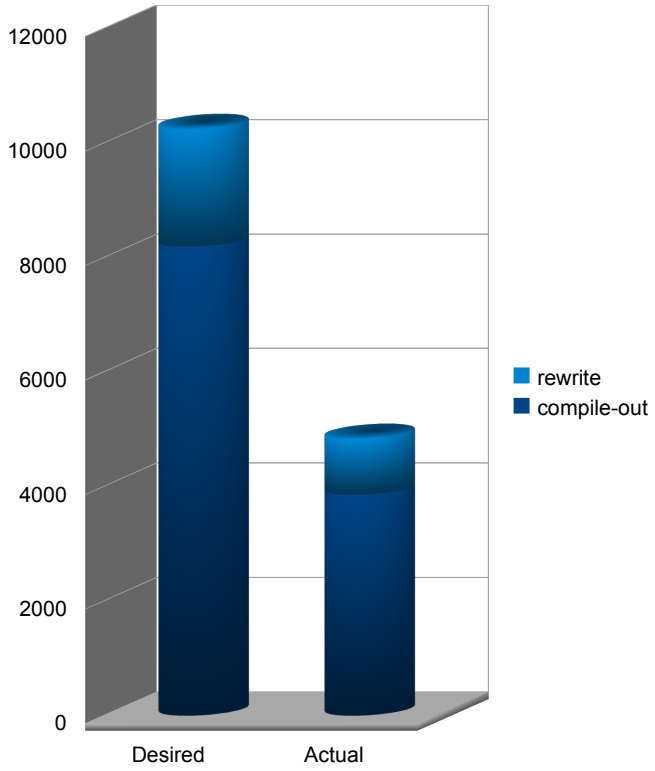
- Tempting target: ProcFS is 20k+ of binary code
  - Userspace depends a lot on ProcFS presence
  - Switching ProcFS off is not an option
- Idea: new config option PROCFS_MINIMAL
  - Expected savings up to 10k
  - Careful selection needed
- Still under way



Analyzing and stripping libraries

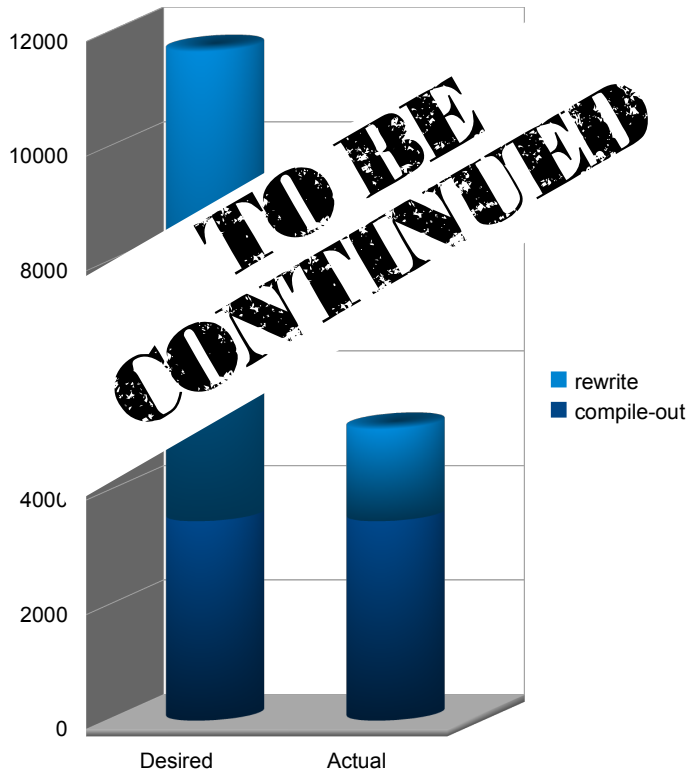
- Many libraries in lib/
  - Most of them are hard to compile out
- Compile out SWIOTLB
  - Make it depend on MMU
  - Saves 4k
- Uninline static functions in CRC32
  - Saves <1k

DONE



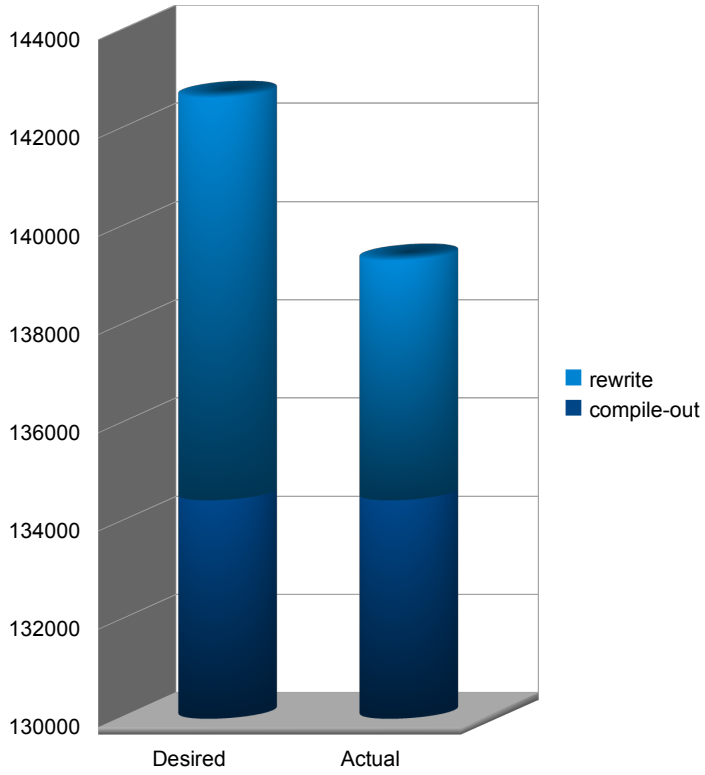
Analyzing kernel system code

- Tempting but complicated
  - kernel/exit.o: 4k .text
  - kernel/signal.o: 9k .text
  - kernel/sys.o: 7k .text
- Less savings but easier
  - kernel/irq/spurious.o: 1k
    - Compile out (-1k)
  - kernel/time/ntp.o: 2k
    - Compile out (-2k)
  - kernel/time/timekeeping.o: 8k
    - Uninline (-1.5k)



Streamlining networking code

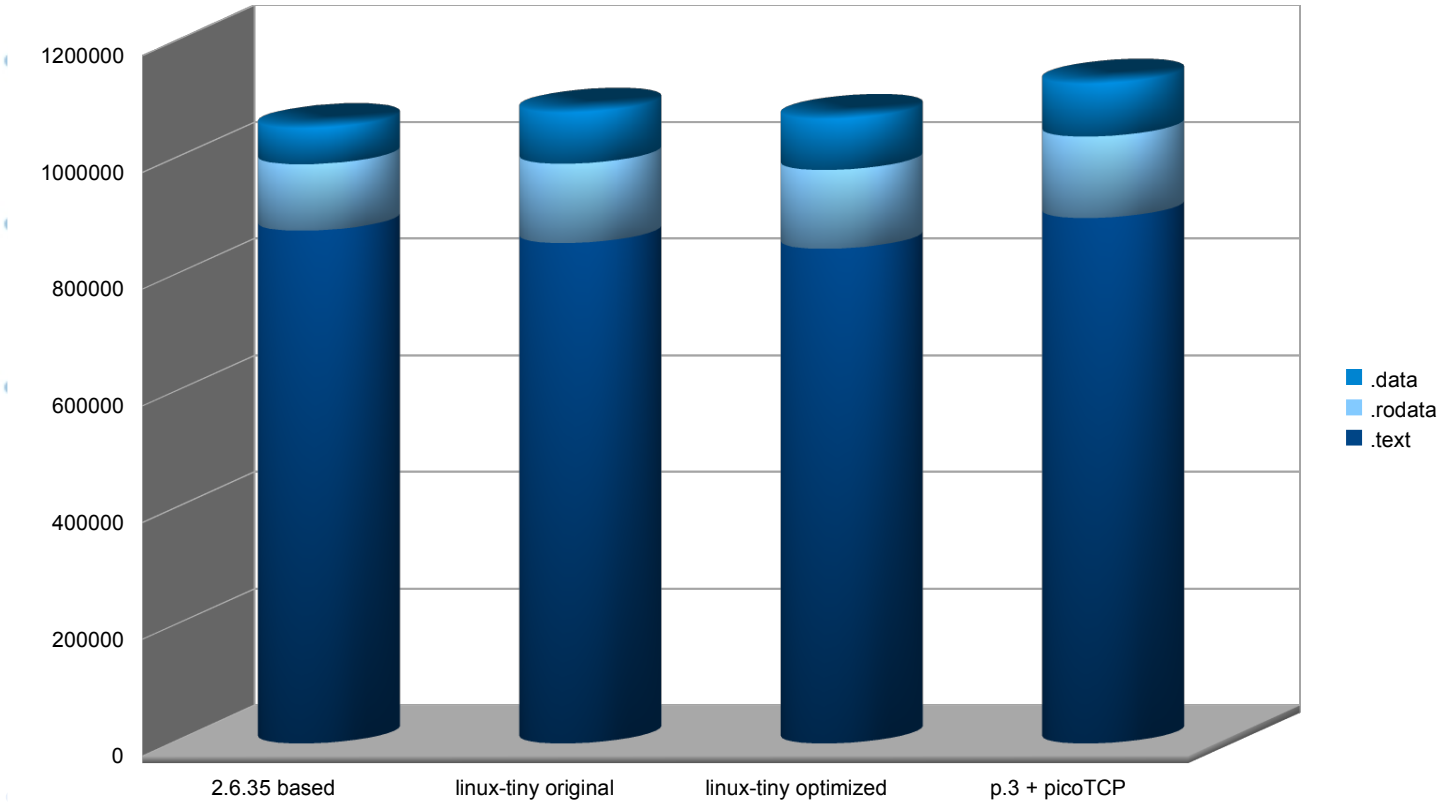
- Do not turn on CONFIG_INET
  - Use picoTCP instead
    - Deserves a separate slide
- Compile out IPv6 stubs
  - 2k saved
- Compile out sysfs stats
  - 3k saved
  - Impact to be estimated



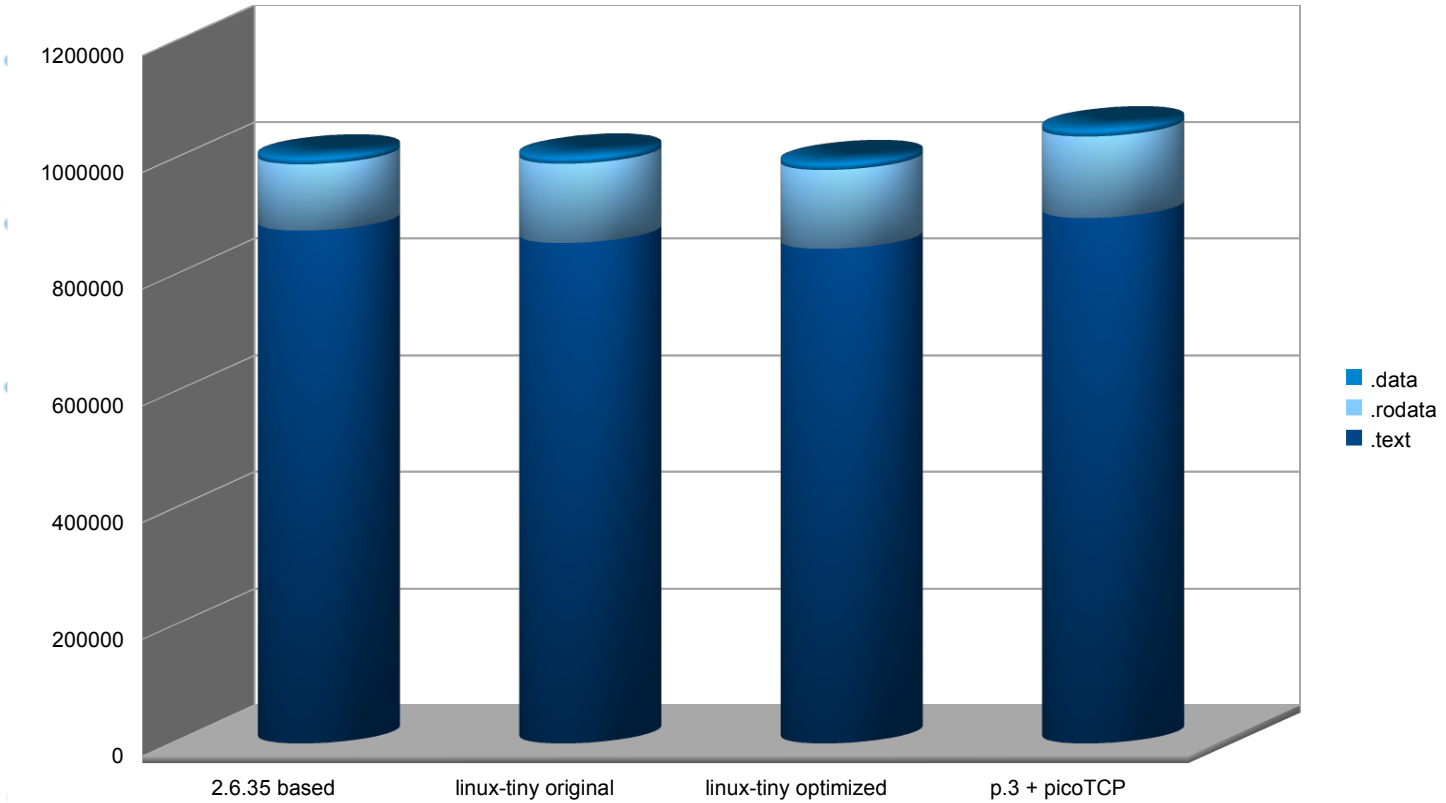
PicoTCP as a kernel module

- PicoTCP was designed to run in userspace
- Efforts were made to make it a kernel module
  - Maxime Vincent, Altran
- We had to redo this
  - No Makefiles publicly available
- Integrated into our version of linux-tiny
  - Only IPv4
  - ~40k binary code added
- Some functionality still missing
  - e. g. no *rtnetlink*

XIP kernel: now and then

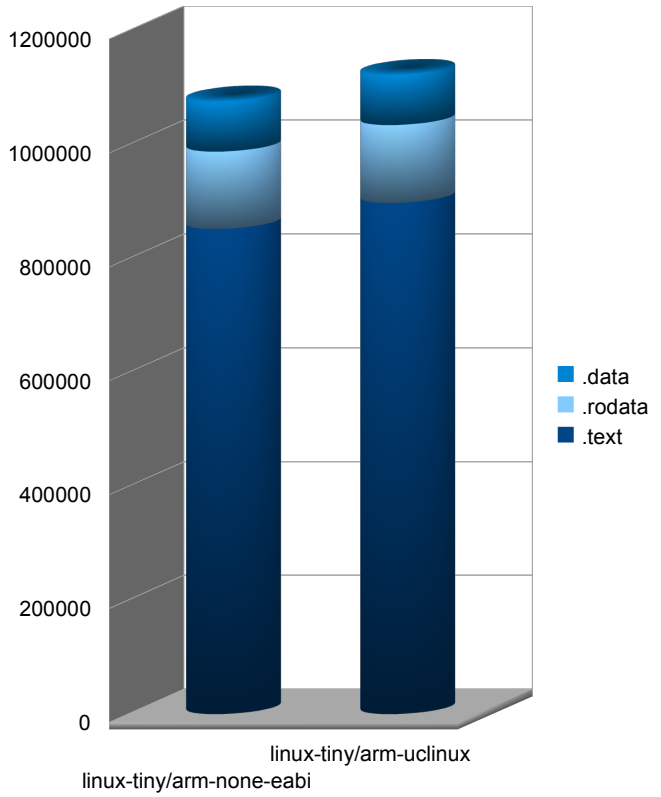


XIP now and then: .data deflated



XIP kernel: toolchain matters!

- Emcraft used arm-uclinuxeabi t/c
  - From CodeSourcery
  - 4.4.1 based
- With linux-tiny, we switched to arm-none-eabi
  - 4.7.4 based
- And saw the improvement



Conclusions

- It is possible to sync up with the mainline and keep the code size down
  - One of the biggest uplifting concerns is not valid
  - Even better results with the new toolchain
- Some size optimizations targeting MCU can be reused by broader audience
  - XIP with compression
- Some performance optimizations for MCU boot-up can be reused by mainstream as well
  - e. g. boot-up time improvements
- There is still a lot to optimize!

Thanks for your attention!

Questions?

mailto: vitaly.wool@softprise.net