RISC-V® Summit

Software "PPA" Metrics

Results from Real-World Security Applications

Joe Circello

Fellow, Chief MCU Core Platform & Security Architect

NXP Semiconductors, N.V.

NXP

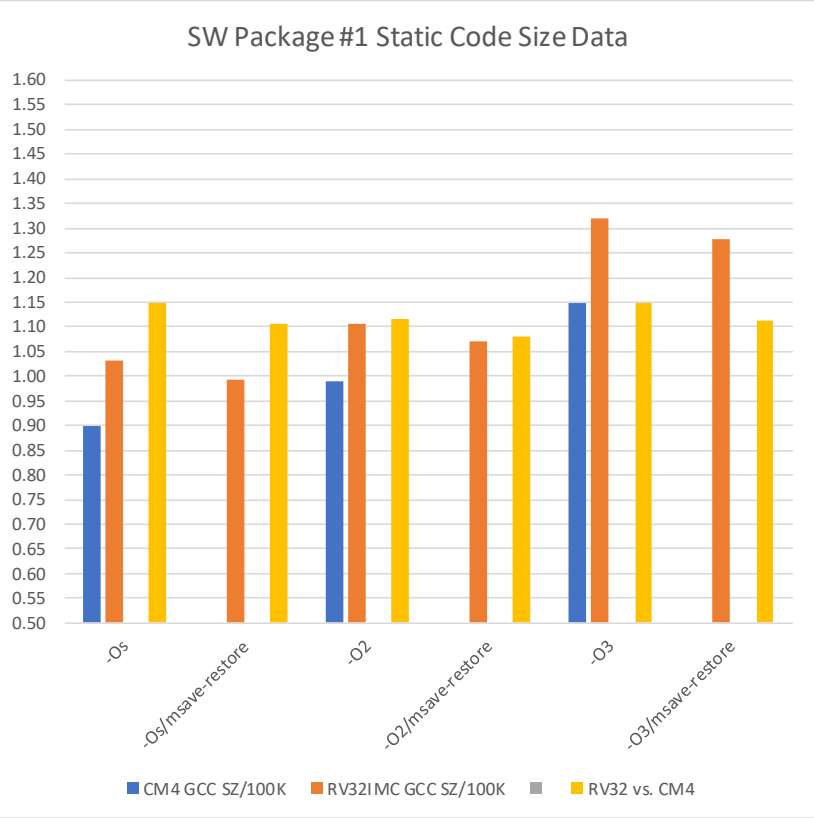
Software Power, Performance, Area (PPA) Metrics

- Tradeoffs between traditional hardware PPA metrics are well-known in SoC design
- In this work, we “redefine” PPA into a software context for microcontroller applications
  - P = relative power in terms of memory (code + data) access rates
  - P = relative performance in terms of execution cycles and dynamic instruction path
  - A = relative static code size
- This work will further limit its focus here to be a *study of relative static code sizes*
  - For constrained MCU compute environments, static code size is a key parameter
    - Non-volatile code memory (flash, ROM) are typically relatively large components and represent a significant amount of dynamic power dissipation
- Applications studied involve “black-box” security subsystems with processing elements
  - Two *production security* software releases: i.MX & “CryptoLib” for MCU devices
    - Original implementations use Arm Cortex-Mx cores as processing elements
      - Retarget to the RISC-V ISA for this study with `-march = RV32IMC`
      - Explore various compiler optimization levels, e.g., -Os, -O2, -O3

More details on the Software Packages

- Software package #1 from i.MX
  - ~100K lines of source C code
  - Arm Cortex-M4 (v7M) GCC vs. RISC-V GCC
  - For both packages:
    - Measuring code sizes of .text + .rodata sections
    - Starting point is the Arm GCC -Os image
  - Compiler optimization levels: -Os, -O2, -O3 plus `–msave-restore` for RISC-V
  - `arm-none-eabi-gcc (GNU Tools for ARM Embedded Processors 6-2017-q2-update) 6.3.1 20170620 (release) [ARM/embedded-6-branch revision 249437]`
  - `riscv32-unknown-elf-gcc (GCC) 7.1.1 20170509 (march=RV32IMC)`
- Software package #2 from CryptoLibrary
  - ~40K lines of source C code
  - Arm Cortex-M4 (v7M) GCC vs. RISC-V GCC and armcc CM4 (v7M) vs. RISC-V GCC
  - `arm-none-eabi-gcc (GNU Tools for ARM Embedded Processors) 4.8.4 20140725 (release) [ARM/embedded-4_8-branch revision 213147]`
  - `ARM Compiler 5.06 update 5 (build 528)`
  - `riscv32-unknown-elf-gcc (GCC) 7.1.1 20170509 (march=RV32IMC)`

Static Code Size Data from Software Package #1



	CM4 GCC	RV32IMC GCC	Relative	CM4 GCC	RV32 GCC
CC Opt flags	[Bytes]	[Bytes]	RV32 vs. CM4	SZ vs -Os	SZ vs -Os
-Os	89856	103048	1.15	1.00	1.00
-Os/msave-restore		99420	1.11		0.96
-O2	99052	110682	1.12	1.10	1.07
-O2/msave-restore		107092	1.08		1.04
-O3	114844	131860	1.15	1.28	1.28
-O3/msave-restore		127860	1.11		1.24

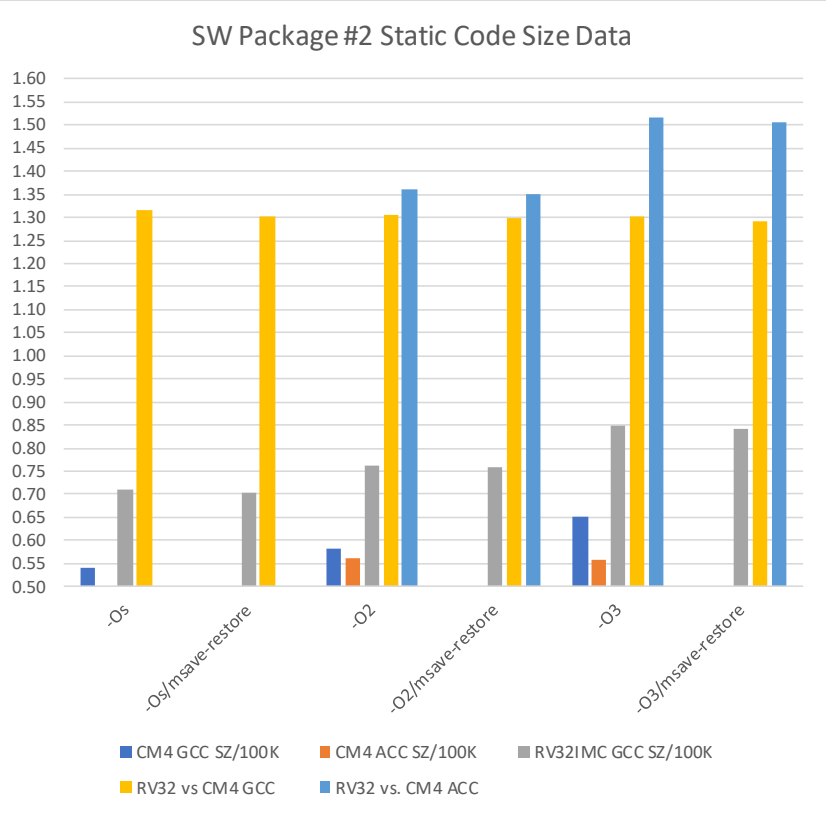
RV32IMC GCC vs. CM4 GCC Code Size Comparison

Relative code size calculated for each compiler option

Smaller relative code size is better

Relative code size range [1.08 – 1.15] at the upper end of expectations

Static Code Size Data from Software Package #2



	CM4 GCC	CM4 ARMCC	RV32IMC	Relative	Relative
CC Opt flags	**[Bytes]**	**[Bytes]**	**[Bytes]**	**RV32 vs. CM4 GCC**	**RV32 vs. CM4 ACC**
-Os	53944		70998	1.32	
-Os/msave-restore			70164	1.30	
-O2	58388	56150	76332	1.31	1.36
-O2/msave-restore			75796	1.30	1.35
-O3	65284	55956	84930	1.30	1.52
-O3/msave-restore			84220	1.29	1.51

RV32IMC GCC vs. {CM4 GCC, CM4 Armcc} Comparisons

- Relative code size calculated for each compiler option
- Armcc options include `-Ospace` and `–execute-only` flags
- *Smaller relative code size is better*

Relative code size range [1.29 – 1.52] *considerably beyond* the upper end of expectations

More Observations from SW #2 and Next Steps

- Attributes of software package #2
  - Highly-modular, relatively-short functions
  - Indirect memory pointer references within data structures
  - APIs with large numbers of parameters
  - Relatively-high data memory access rate per instruction
  - Code written using “secure coding” guidelines
  - Side-by-side review of compiler outputs show similar instruction sequences
    - Inherent differences in instruction lengths seem to account for most of size
  - Belief that these factors contribute to (much) larger-than-expected RV32IMC size
- Next steps…
  - Continue static code analysis in other “production code” application areas
  - Generate runtime analysis metrics for Power and Performance
    - Memory access rates per instruction (instruction fetches, data loads & stores)
    - Stack memory footprint
- Acknowledgements: Antoine Dambre (SW #1) and Pim Vullers (SW #2)

SECURE CONNECTIONS
FOR A SMARTER WORLD