Fast Forward for RISC-V in Space

The Coyote IP core

RISC-V in Space – Workshop on 14 December 2022

Roland Weigand, Felix Siegle, ESA

ESA UNCLASSIFIED – For Official Use

Why...

Why RISC-V?

Why not SPARC forever?

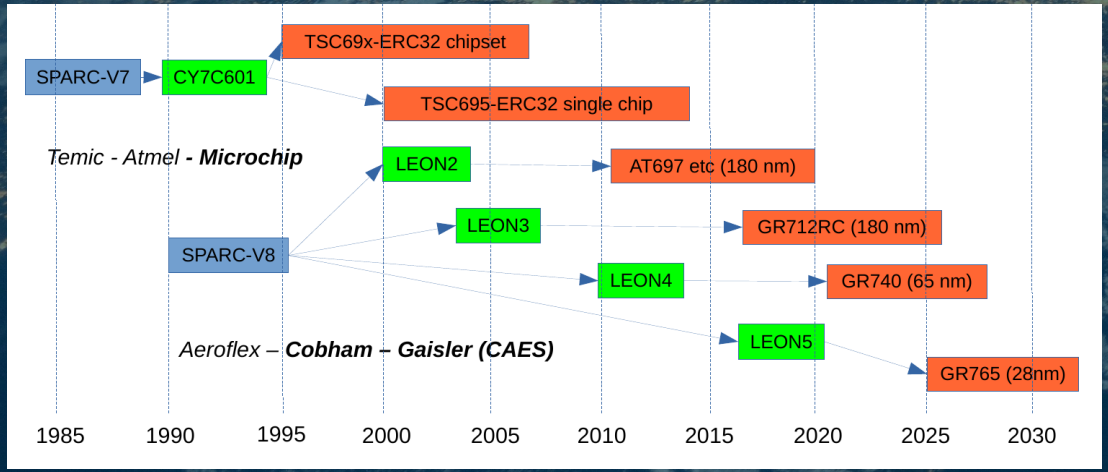
Why not ...?

→ THE EUROPEAN SPACE AGENCY
ESA UNCLASSIFIED – For Official Use

SPARC in space – 25 years – 5 generations

- ERC32 – Based on a commercial SPARC-V7 IP-core, TSC69x series
- LEON2 SPARC V-8 IP-core owned by ESA → AT697F
- LEON3 IP-core owned by CAES (aka Cobham Gaisler) → GR712RC
- LEON4 IP-core, introducing L2 cache → GR740, QML-V in 2022
- LEON5, introducing dual-issue → GR765, under development

moreover, LEON IP is on many other ASIC and FPGA



SPARC Compliant SCD V8

SPARC is in all our missions

1000s of flight parts

A SUCCESS STORY

mature SW ecosystem

demand in space continues

WHY SHOULD WE CHANGE?

obsolete in the commercial world

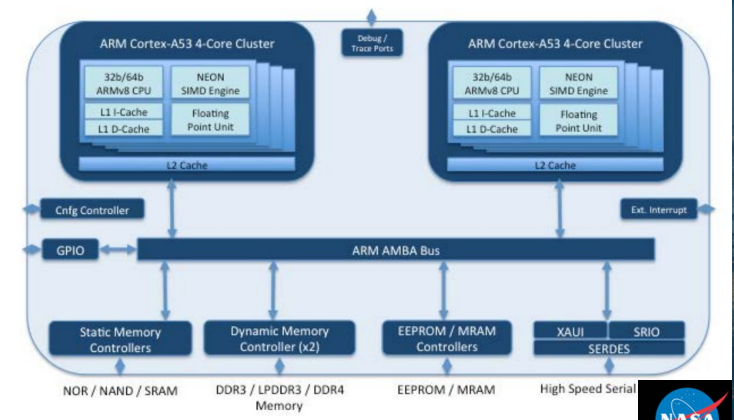
lack of community backing

no more compiler fixes

lack of developers

→ THE EUROPEAN SPACE AGENCY
ESA UNCLASSIFIED – For Official Use

What about a main-stream proprietary ISA?



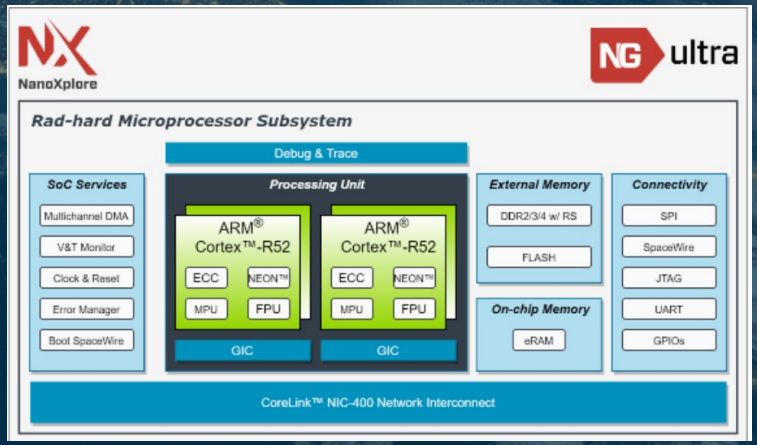
HPSC "Chiplet" Reference Design

Advantages:

- High quality / maturity of the IP
- Mature SW ecosystem
- IP and support from one hand
- Large developer community

Drawbacks:

- Single source IP provider
- No modifications allowed (hardening)
- ¨one hand¨ scattered over many sites
- SW ecosystem not suitable for space
- Cost? – but quality is never for free



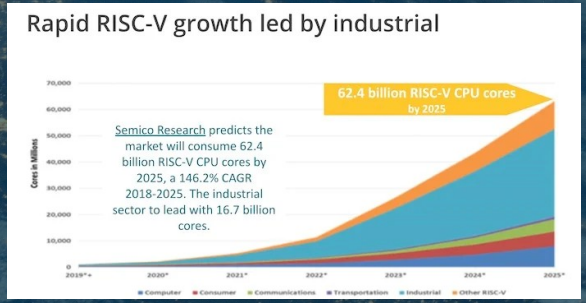
→ THE EUROPEAN SPACE AGENCY
ESA UNCLASSIFIED – For Official Use

Why RISC-V?

Advantages:

- Open ISA – just like SPARC – but RISC-V has active standardization
- Rapidly growing, **multiple IP sources**: commercial, open-source, free
- **Highly extensible**: ISA covers low-end to high-performance applications
- Overcomes annoying ¨features¨ of SPARC (delay slots, register windows)
- Rapidly growing developer community at all levels (industry, academia)

Name	Description	Version	Status	Instruction Count
RV32I	Base Integer Instruction Set - 32-bit	2.1	Frozen	49
RV32E	Base Integer Instruction Set (embedded) - 32-bit, 16 registers	1.9	Open	Same as RV32I
RV64I	Base Integer Instruction Set - 64-bit	2.0	Frozen	14
RV128I	Base Integer Instruction Set - 128-bit	1.7	Open	14
	Extension			
M	Standard Extension for Integer Multiplication and Division	2.0	Frozen	8
A	Standard Extension for Atomic Instructions	2.0	Frozen	11
F	Standard Extension for Single-Precision Floating-Point	2.0	Frozen	25
D	Standard Extension for Double-Precision Floating-Point	2.0	Frozen	25
G	Shorthand for the base and above extensions	n/a	n/a	n/a
Q	Standard Extension for Quad-Precision Floating-Point	2.0	Frozen	27
L	Standard Extension for Decimal Floating-Point	0.0	Open	Undefined Yet
C	Standard Extension for Compressed Instructions	2.0	Frozen	36
B	Standard Extension for Bit Manipulation	0.90	Open	42
J	Standard Extension for Dynamically Translated Languages	0.0	Open	Undefined Yet
T	Standard Extension for Transactional Memory	0.0	Open	Undefined Yet
P	Standard Extension for Packed-SIMD Instructions	0.1	Open	Undefined Yet
V	Standard Extension for Vector Operations	0.7	Open	186
N	Standard Extension for User-Level Interrupts	1.1	Open	3
H	Standard Extension for Hypervisor	0.0	Open	2
S	Standard Extension for Supervisor-level Instructions	1.12	Open	7



Drawbacks:

- Limited maturity of some IP
- Long process for extension ratification
- Not clear if all extensions will ¨survive¨
- Custom extensions are risky: SW support!

→ THE EUROPEAN SPACE AGENCY
ESA UNCLASSIFIED – For Official Use

Prior RISC-V R&D – First Steps at ESA

- Limited RISC-V efforts at ESA until 2021 (few 100 k€):
  - small studies, benchmarking, internal technical work, trainees and PhD
- Open Instruction Set Architectures (ISA) in Space – ADCSS 2019
  http://microelectronics.esa.int/papers/OpenSourceISA-RolandWeigandPub-2019-11-13.pdf
- RISC-V First Steps Into Space – ADCSS 2021
  http://microelectronics.esa.int/papers/RISCV-ADCSS-2021-11-18.pdf
  (video on ADCSS site)
- Several EC H2020 grants running from 2019 – 2022
  - SELENE, De-RISC (not only targeting space)
- Lobbying mostly unsuccessful
  - People did not know how to spell RISC-V

→ THE EUROPEAN SPACE AGENCY
ESA UNCLASSIFIED – For Official Use

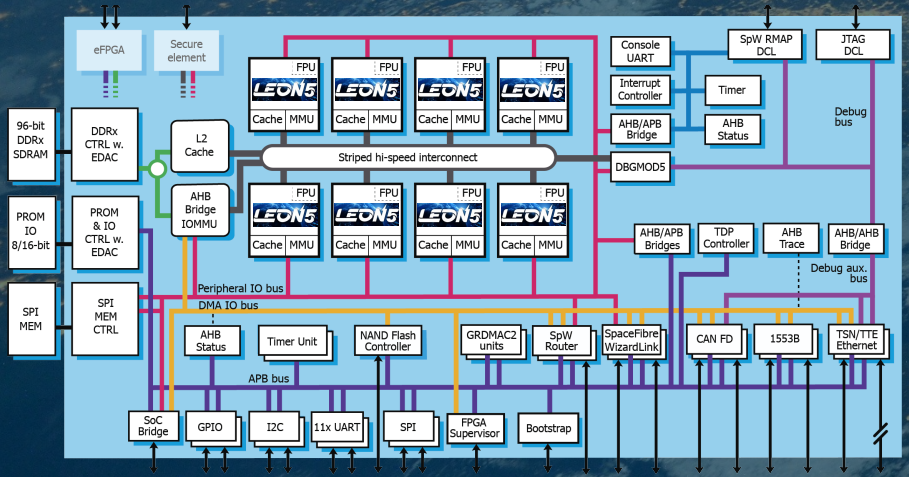
2021/22: RISC-V@ESA in Fast Forward Mode

- Concept study for GR7xV HPC started in ARTES in 2021
  - Target 7 nm – technology not available for space
- 2022 started with a ¨shoestring¨ ITT in TDE: 450k€
  - Can we do RISC-V prototypes in 22-28 nm technology?
- Synergy with GR765 (originally planned as 8-core LEON5 only)
  - → RISC-V is hitch-hiking on SPARC, saving cost...
  - Only CPU cores are duplicated, but cache RAM and SoC shared
  - Facilitates transition for users: one chip, one board, two modes
- GR765 phase 2 released with ARTES funding also in 2022
- Today ~ 4 M€ public funding committed into GR765 / GR7xV
  - complemented by significant co-funding
  - Prototypes funded, flight parts (ROM cost 4.5 M€) TBC

→ THE EUROPEAN SPACE AGENCY
ESA UNCLASSIFIED – For Official Use

GR765 – transition from SPARC …

- GR765 Phase 1 : Multi-Core LEON5FT (SPARC only) Microprocessor Development, started 03/2021
- GR765 Phase 2 : Layout, package design, manufacturing (ST 28nm FDSOI), started 07/2022
  - GR765 was first planned as a pure SPARC chip
  - No ESA ITT – initiated by CG responding to market demand for higher performance and enhanced interfaces

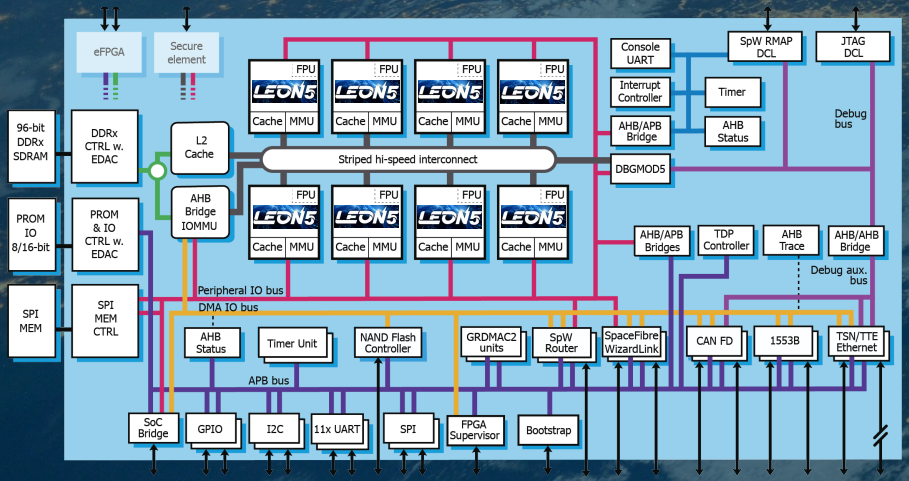


→ THE EUROPEAN SPACE AGENCY
ESA UNCLASSIFIED – For Official Use

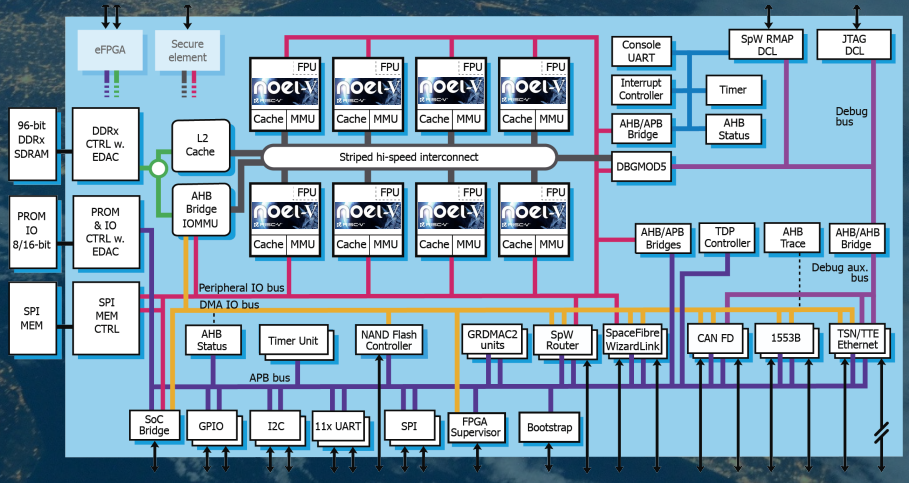
GR765 – transition from SPARC … to RISC-V

- GR765 Phase 1 : Multi-Core LEON5FT (SPARC only) Microprocessor Development, started 03/2021
- GR765 Phase 2 : Prototypes in ST 28nm FDSOI, started 07/2022
- ITT “RISC-V Microprocessor Prototypes”, started 07/2022

→ **RISC-V is “hitch-hiking” on the GR765 design and product concept which was already in progress**



GR765 in SPARC/LEON5 mode

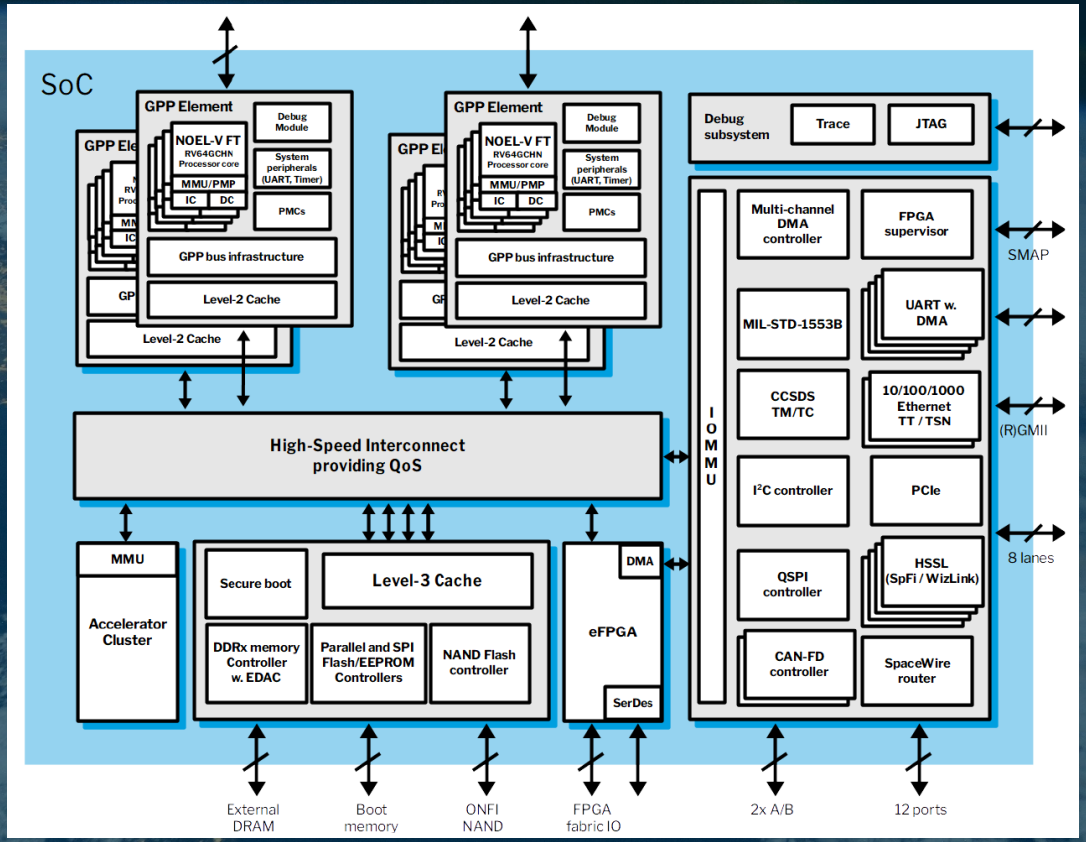


GR765 in RISC-V/NOEL-V mode

→ THE EUROPEAN SPACE AGENCY
ESA UNCLASSIFIED – For Official Use

GR7xV

- 4x4 core RISC-V Processor mode
- Target 7 nm (UDSM)
- Level 3 cache
- Accelerator, eFPGA
- Chiplet partitioning TBD
- Phase 1 (Architectural Design) started Q3/2021 (ARTES, 1135 k€)
- Manufacturing date TBC, pending UDSM space ASIC platform
  - ~70 M€ subscribed for ”EEE sovereignty”
  - Split of cake is unclear
  - Foundry NDA difficult



→ THE EUROPEAN SPACE AGENCY
ESA UNCLASSIFIED – For Official Use

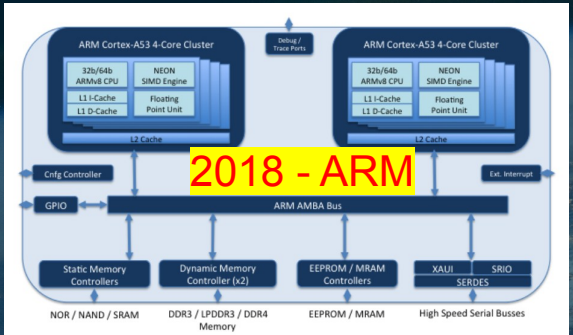
RISC-V ancillary needs (ecosystem)

- Associated IP developments
  - Vector / SIMD extensions → Development, validation, compiler DSP libraries
- Hypervisor (XtratuM porting done in H2020-DeRISC) → qualification
- Operating systems with drivers and boot SW (RTEMS, Linux, PikeOS…) → porting and qualification
- Compilers : can we rely on community?
- AI inference tools
- Benchmarks (OBPMark, AI evaluation)
- Debug and trace tools
- Predictability / timing analysis
- Parallel programming tools (OpenMP...)
- Simulators / Emulators → timing accuracy vs performance trade-off?
  - Transactional (e.g. SystemC) for SoC design space exploration
  - Instruction Set Simulator (ISS) and FPGA emulator for Software Validation (QEMU)
- Security addenda: Secure Boot (root of trust), Secure Islands (enhanced MMU, IOMMU), STM/STC

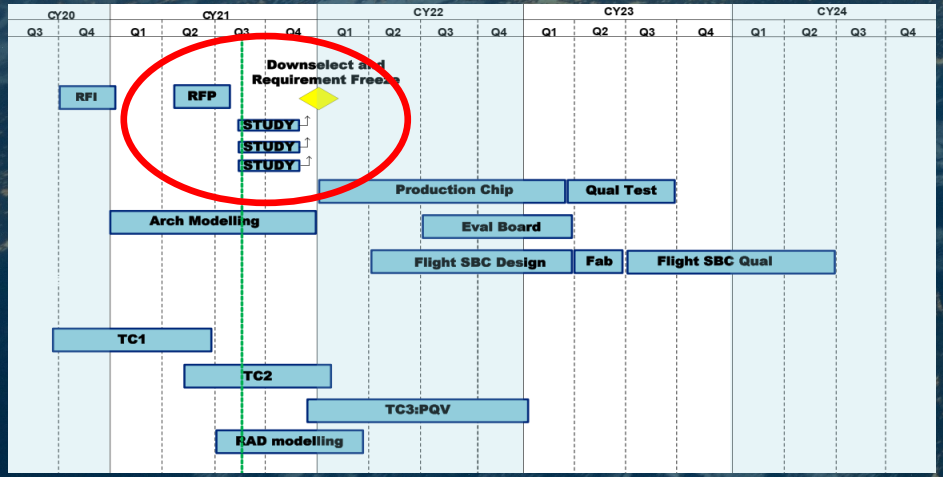
planning and fundraising for these activities is in progress

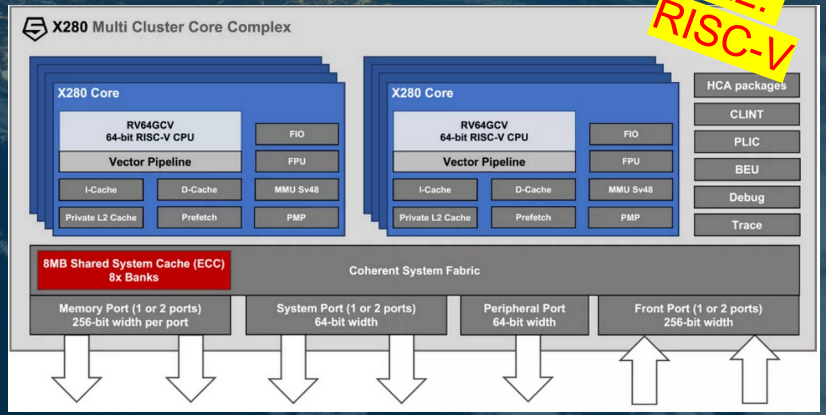
→ THE EUROPEAN SPACE AGENCY
ESA UNCLASSIFIED – For Official Use

NASA High Performance Space Computing (HPSC)



- 2018 plans: 32 nm SOI, ARM Cortex A53
  [W. Powell, RadHard Electr Techn 2018]
- 2021 refurbishment: 22 nm (or below), ISA open
  [J. Butler, Space Comp Conf 2021]
- 2021: 3 parallel architecture studies: trade-off ARM and RISC-V?
- 2022/08: awarded contract to Microchip, using SiFive RISC-V cores
  https://www.techspot.com/news/95911-sifive-risc-v-cores-microchip-processors-power-nasa.html



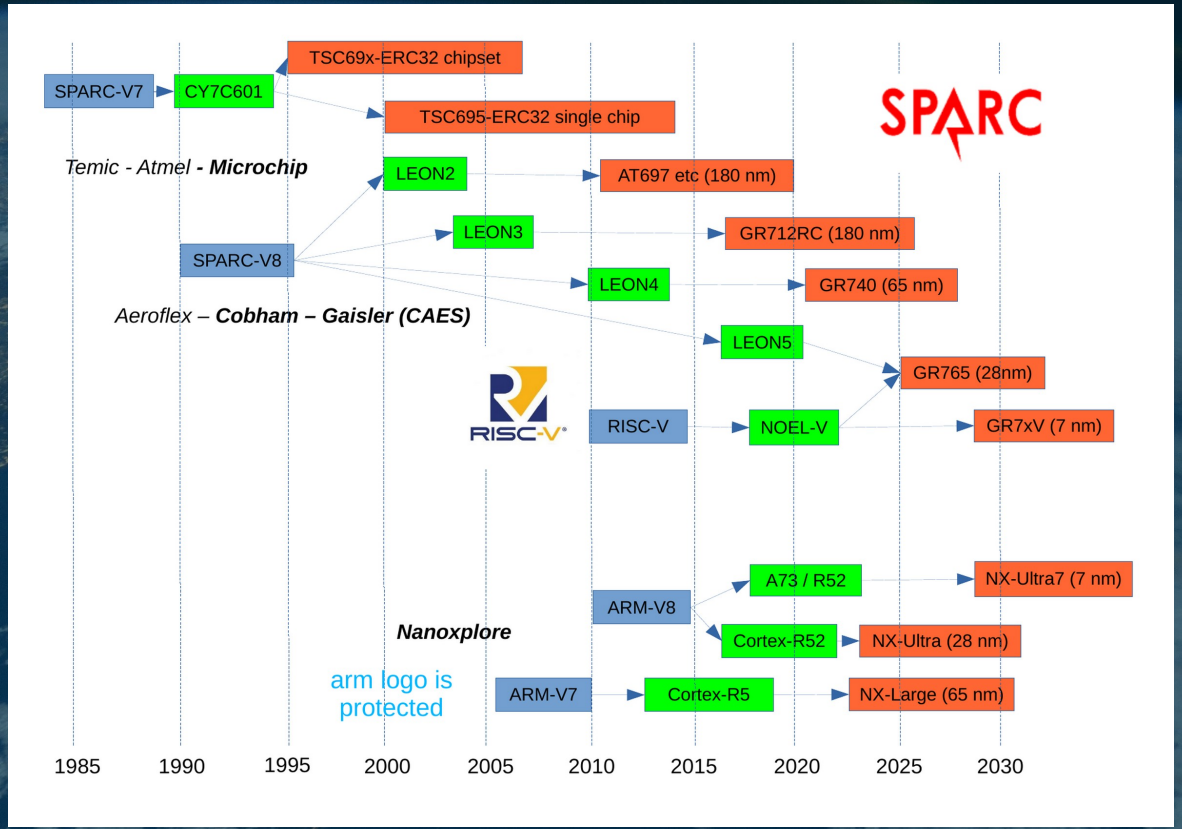


→ THE EUROPEAN SPACE AGENCY
ESA UNCLASSIFIED – For Official Use

Space Micropocessor Roadmaps in Europe

- Almost 30 years of SPARC
- LEON chips by Microchip and Cobham Gaisler (CG)
- CG transition to RISC-V
- RISC-V on Microchip PolarFire-SoC FPGA
- NASA HPSC-new opted RISC-V (SiFive/Microchip)

- NX roadmap: ARM
- ARM also on Xilinx FPGA

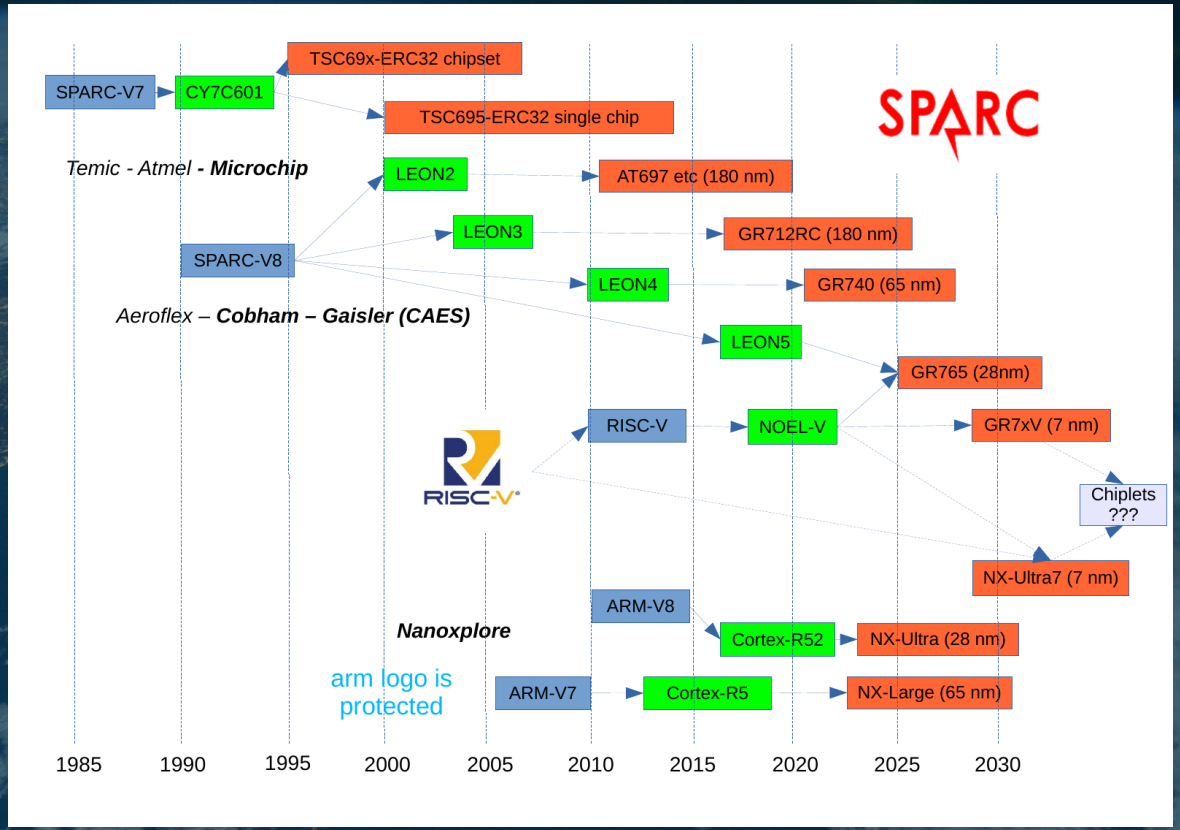


→ THE EUROPEAN SPACE AGENCY
ESA UNCLASSIFIED – For Official Use

Space Micropocessor joint European Roadmap

- Almost 30 years of SPARC
- LEON chips by Microchip and Cobham Gaisler (CG)
- CG transition to RISC-V
- RISC-V on Microchip PolarFire-SoC FPGA
- NASA HPSC-new opted RISC-V (SiFive/Microchip)

- Harmonizing the ISA to RISC-V between European processor and FPGA vendors would be highly desirable.



→ THE EUROPEAN SPACE AGENCY
ESA UNCLASSIFIED – For Official Use

Acknowledgements

We are at the beginning of our RISC-V adventure ....

… acknowledgements go to all colleagues (*) and people in industry which help

(*) in particular in TEC-EDD, TEC-SWF, TEC-SWT, TIA-TTS

- Reviewing specs
- Providing use cases
- Benchmarking
- Managing the contracts

- … promoting RISC-V, lobbying, fundraising

→ THE EUROPEAN SPACE AGENCY
ESA UNCLASSIFIED – For Official Use

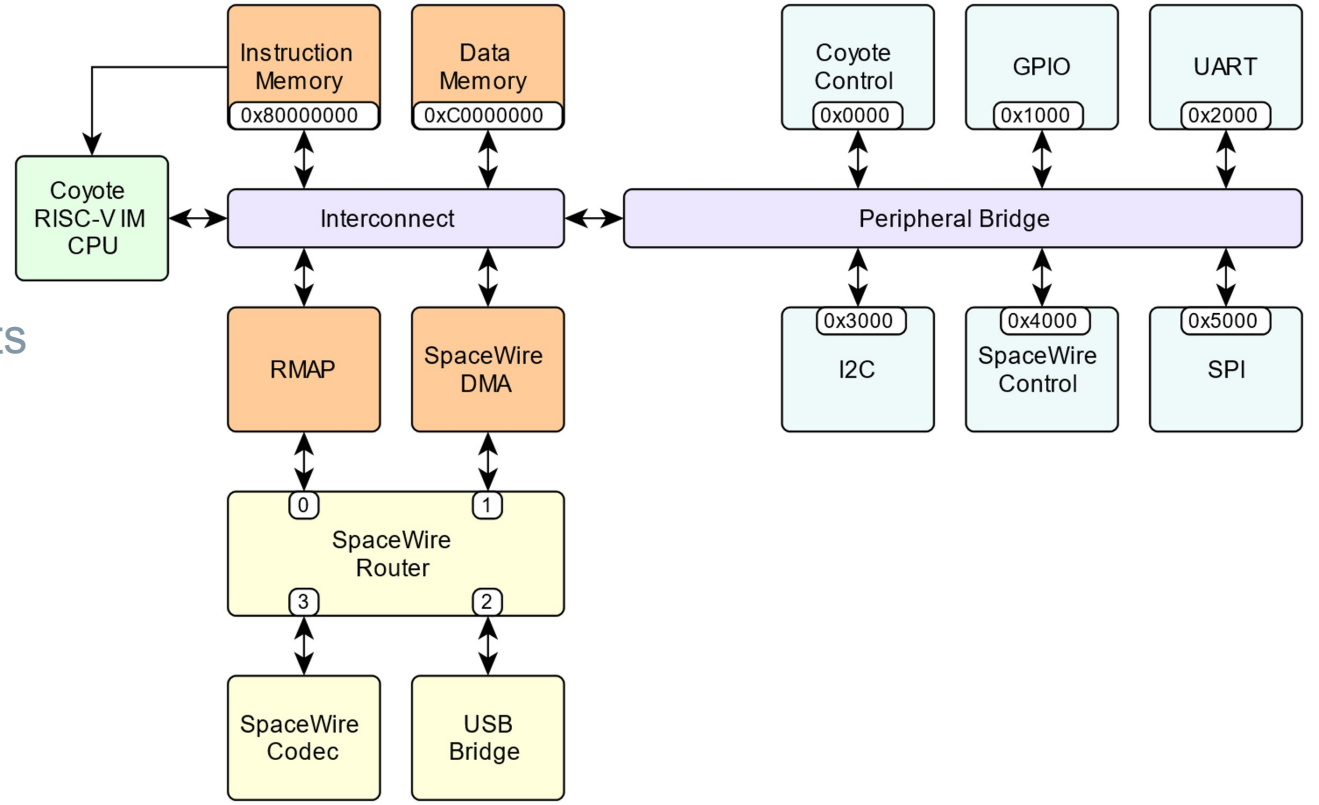
Homebrew RISC-V on BRAVE-Large

Felix Siegle, TEC-EDD
RISC-V in Space Workshop, 14/12/2022

ESA UNCLASSIFIED - For ESA Official Use Only

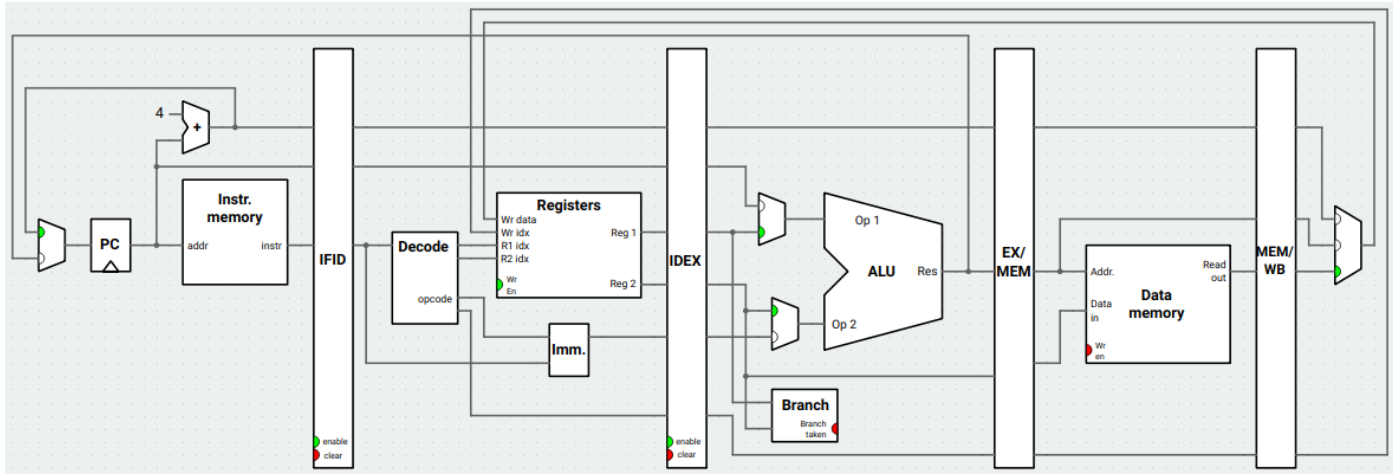
”Coyote” System-on-Chip

- **Hobby Covid-19 project**
- Motivation:
  - Learn more about RISC-V
  - Learn SystemVerilog (in addition to VHDL)
  - Learn more about low-level software aspects (linker scripts, bootloader etc.)
- Fully functional but minimalistic System-on-Chip:
  - Classic 5-stage pipeline RISC-V CPU implementing I and M extensions **(RV32IM)**
  - On-chip instruction and data memory
  - Avalon interconnect
  - Low-speed peripherals: GPIO, UART, SPI, I2C
  - SpaceWire sub-system: Codec, Router, RMAP Target, DMA Engine
  - USB2 <> SpaceWire bridge



Coyote System-on-Chip

- Design is separated into two distinct parts:
  - “Inner” SoC includes CPU, interconnect, low-speed peripherals, DMA, RMAP, control logic blocks (**SystemVerilog**)
  - “Outer” design includes SpaceWire codec, SpaceWire router, USB interface (**VHDL**)
- CPU is implemented using **5 stages** and reaches good performance (**1.8 CoreMark/MHz**, up to 100 MHz on Spartan-6)



© https://hackmd.io/@WeiCheng14159/rkUifs2Hw

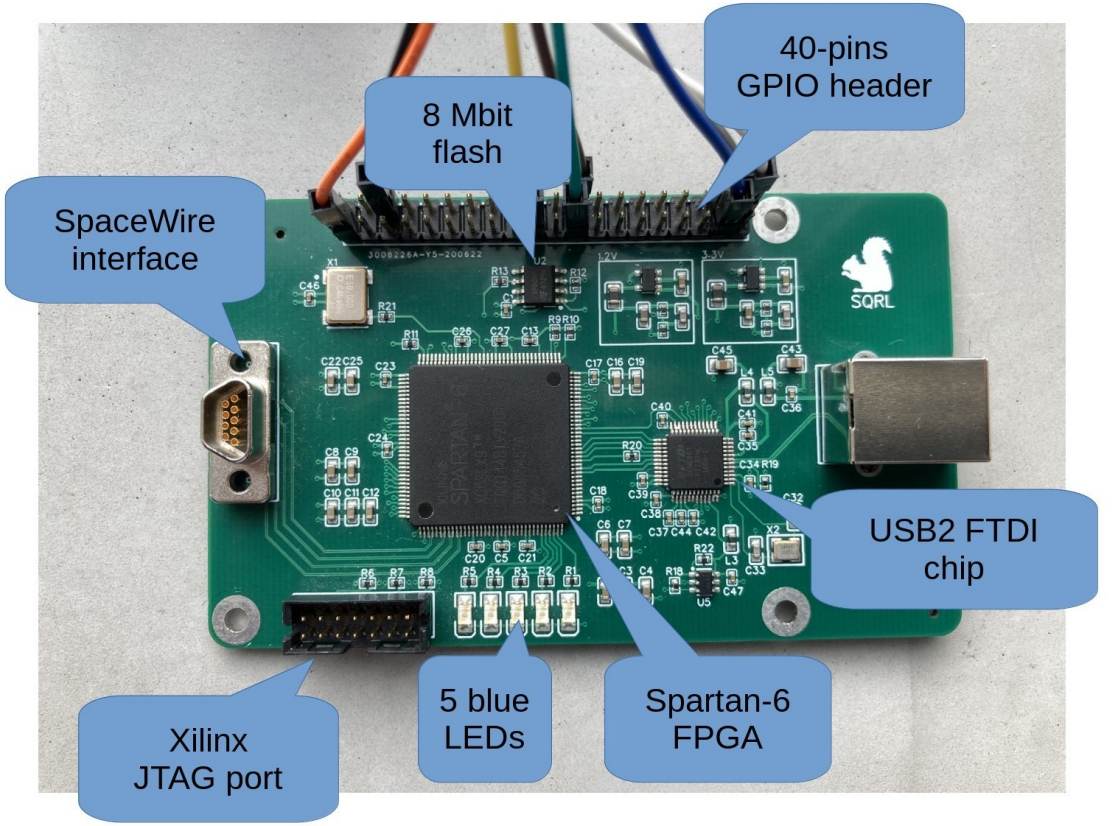
RV32I Base Instruction Set

imm[31:12]			rd	0110111	LUI
imm[31:12]			rd	0010111	AUIPC
imm[20|10:1|11|19:12]			rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
fm | pred | succ		rs1	000	rd	0001111	FENCE
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK

+ HW MUL/DIV instructions (M extension)

Poor man’s SpaceWire brick

- Small, self-made FPGA board based on Spartan-6 FPGA
- Produced in China for ~ $50 incl. all parts
- Includes Raspberry Pi-compatible 40-pins header
- Implements the whole SoC but is only used as USB-SpaceWire bridge in this project
- **100 MHz clock**
- Can sustain full bi-directional SpaceWire 200 Mbps data rate
- Signal integrity of SpaceWire link checked with SpaceWire sniffer / scope

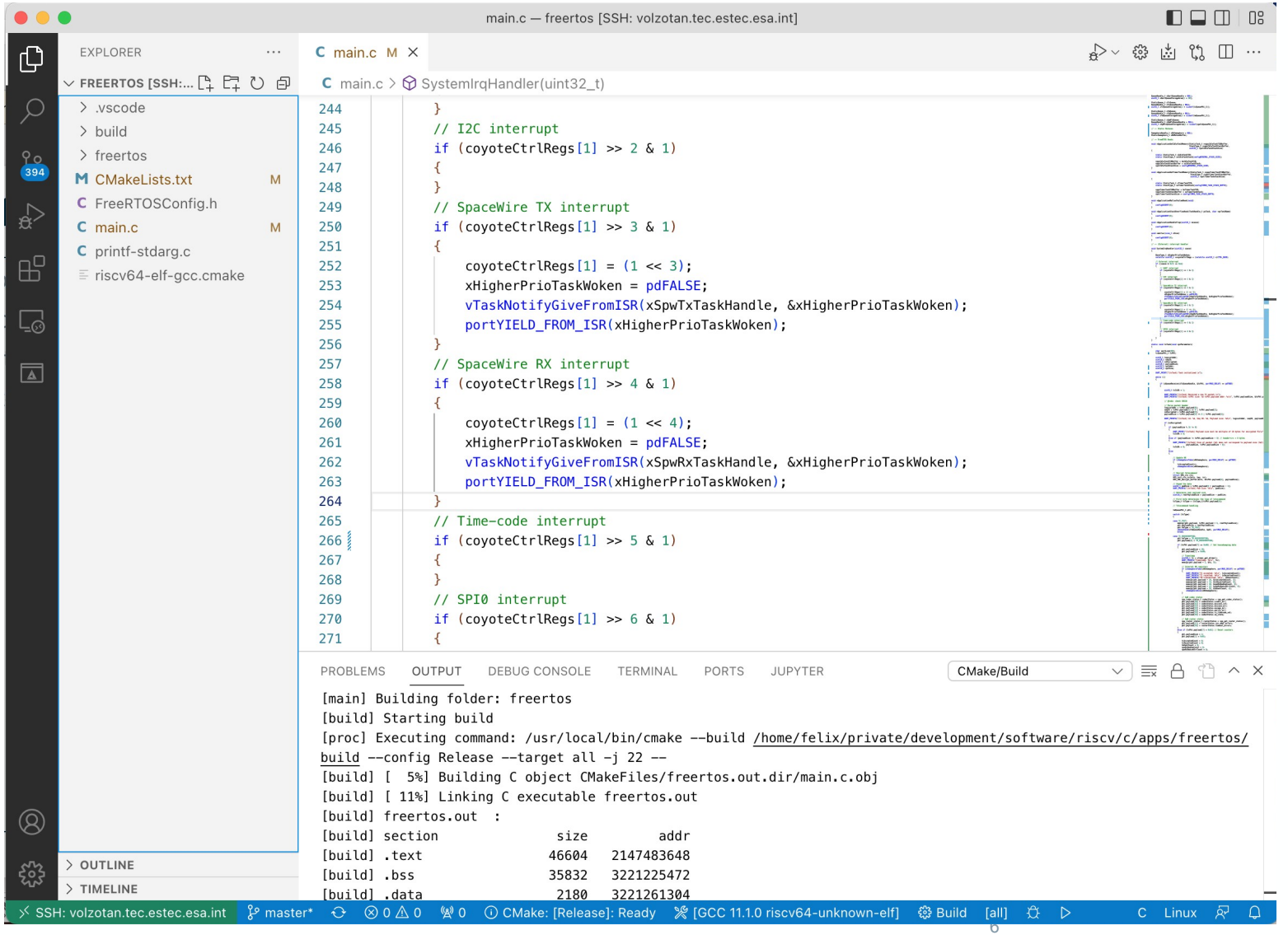


Verification / validation

- All IP cores are “kind of” verified in simulation
- SystemVerilog testbench allows execution of binaries on CPU in simulation
- Several programs written in assembler / C for hardware tests (SpaceWire, interrupts etc.)
- A few self-checking programs executed successfully:
  - CoreMark
  - AES-256 encryption/decryption
- Executes the RISC-V compliance suite successfully (I & M extensions)
- Runs FreeRTOS successfully, described in more detail later
- SpaceWire codec successfully validated with STAR-Dundee conformance tester
- RMAP target checked for correctness by using RMAP library from STAR-Dundee

Software environment

- Software can be written in Assembler or C/C++ (and others)
- Open source RISC-V GCC/Clang available
- Projects are managed with CMake
- Integrated everything into **Visual Studio Code** ☾ nice for C, assembler, python, SV, VHDL…



Software drivers

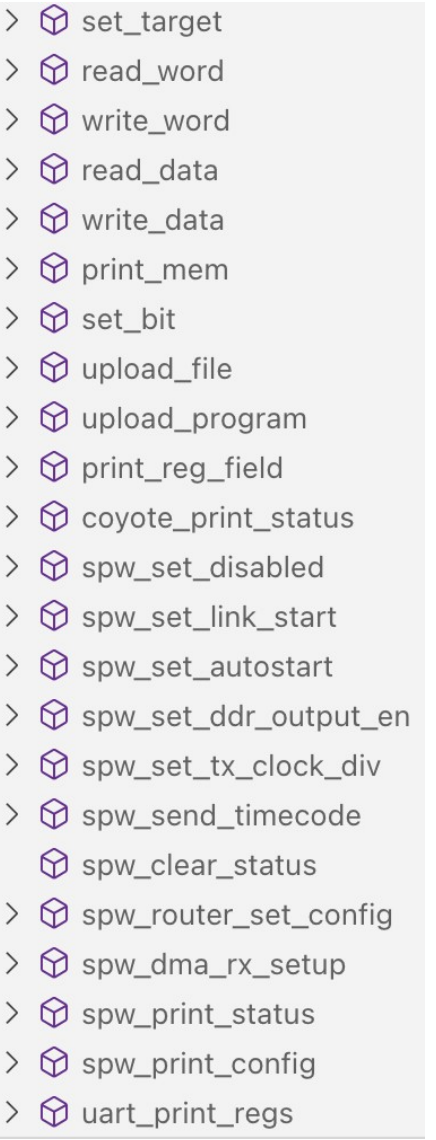
- UART
  Put char, get char, print string, set baudrate
- SPI
  Init, setup transfer, run transfer, chip select, send data
- I2C
  Init, start, stop, write
- GPIO
  Get values, set values, set direction, set interrupts, write single value
- System Timer
  Get/clear MTIME, set time compare value, set/get pre-scaler, wait for amount of ticks
- SpaceWire
  See screenshot
- Flash Memory
  Read, Page Program, Sector Erase, Write enable/disable

SpaceWire driver API

- spw_init()
- spw_codec_set_div(uint8_t, uint8_t)
- spw_codec_set_autostart(uint8_t)
- spw_codec_set_start(uint8_t)
- spw_codec_set_disabled(uint8_t)
- spw_codec_en_timecode_int(uint8_t)
- spw_codec_set_tx_timecode_val(uint8_t)
- spw_codec_tx_tick_in()
- spw_codec_clr_errors()
- spw_router_set_timeout(uint16_t)
- spw_router_set_timeout_en(uint8_t)
- spw_router_set_allow_self_addr(uint8_t)
- spw_dma_write(void *, uint32_t, uint8_t)
- spw_dma_read(void *, uint32_t, uint8_t)
- spw_dma_set_tx_addr(uint32_t, uint8_t)
- spw_dma_en_tx_int(uint8_t)
- spw_dma_en_rx_int(uint8_t)
- spw_get_codec_status()
- spw_get_codec_ctrl()
- spw_get_router_status()
- spw_get_router_ctrl()
- spw_get_dma_status()
- spw_dma_clr_errors()
- spw_dma_cancel_rx()
- spw_router_clr_errors()
- spw_get_rmap_status()

Poor man’s GRMON

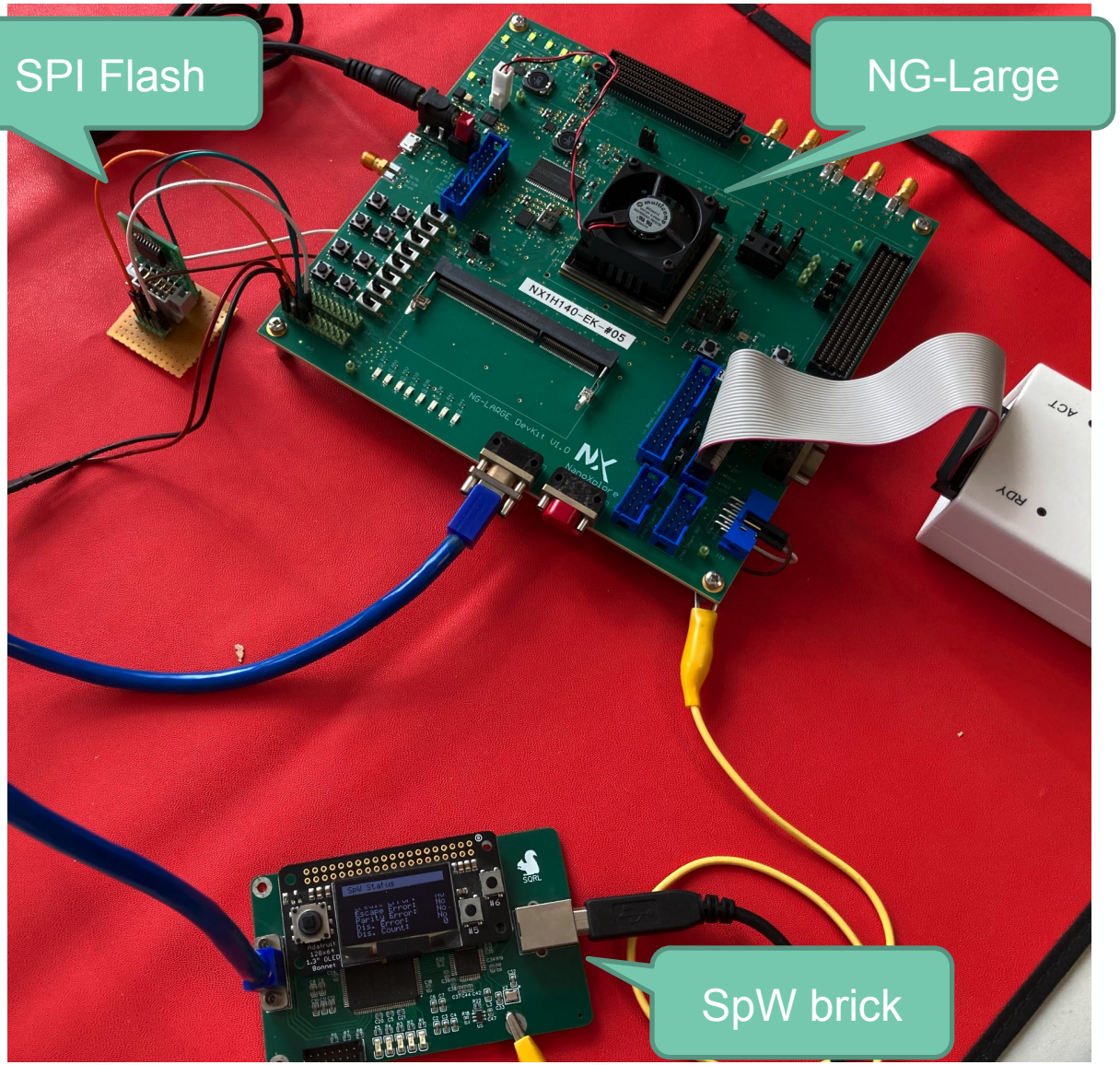
- Need some way to communicate with SoC
- Implemented a USB SpaceWire C library based on libusb / libftdi and a RMAP library to read/write registers
- Works on Linux, Mac, Windows
- To simplify writing test scripts etc., I also created a Python module, that includes functions such as: rmap_write_word, rmap_read_data, raw_send_and_receive
- On top of that, some Python functions add convenicene/debugging functionality, e.g., print a range in memory, upload a program, print configuration registers



Python library for convenience/ debugging functions

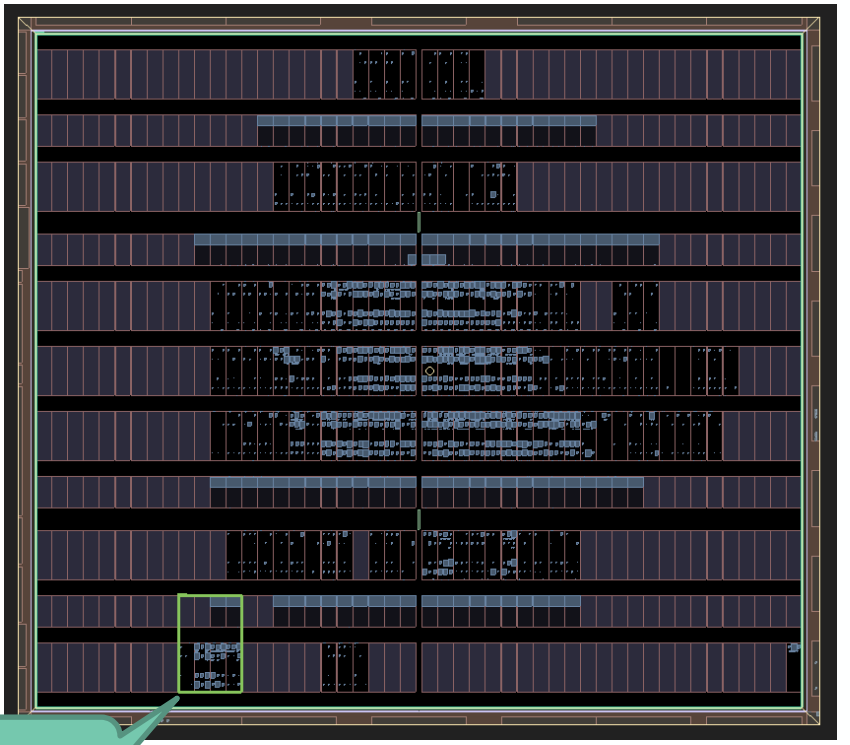
NG-Large port

- Same SoC as the one on development board, including the following IPs:
  - UART, I2C, SPI
  - SpaceWire (Codec, Router, RMAP, DMA)
- 128 kB on-chip instruction memory
- 64 kB on-chip data memory
- Targeted speed:
  - **CPU: 25 MHz**
  - **SpaceWire: 50 Mbps**
- Reference: Cyclone V / Spartan 6
  - CPU: 80 – 100 MHz
  - SpaceWire: > 200 Mbps



NG-Large results

- SystemVerilog part had to be translated to Verilog
- Tiny core, uses only ~5-10% of the resources
- Large amount of BRAMs because of missing byte-enables and ECC scheme
- Needed to rewrite / pipeline the interconnect to achieve better timing
- Couldn’t figure out some of the macros, e.g. register files
- NG-Large Dev board has AC caps in SpaceWire lines…



Domain		Frequency	
Source	Target	Required	Maximum
Input	clk_50MHz (Falling)	–	–
Input	clk_50MHz (Rising)	–	–
Input	sys_clk (Rising)	–	–
clk_50MHz (Rising)	Output	–	–
clk_50MHz (Falling)	clk_50MHz (Rising)	–	–
clk_50MHz (Rising)	clk_50MHz (Rising)	50.000 MHz	58.065 MHz
clk_50MHz (Rising)	sys_clk (Rising)	–	–
sys_clk (Rising)	Output	–	–
sys_clk (Rising)	clk_50MHz (Rising)	–	–
sys_clk (Rising)	sys_clk (Rising)	25.000 MHz	27.635 MHz

4-LUT	DFF	XLUT	1 - bit Carry	Register file block	Cross domain clock	Clock Buffer	Clock switch	Digital signal processor	Memory block	WFG	PLL
10759/129024 (9%)	4784/129024 (4%)	0/8064 (0%)	1537/32256 (5%)	0/672 (0%)	0/672 (0%)	0	0/1344 (0%)	4/384 (2%)	98/192 (52%)	2/40 (5%)	1/4 (25%)

NG-Large Coyote “validation”

- Executed a couple of tests:
  - Full memory check through RMAP
  - Simple UART test
  - All RISC-V compliance tests
  - SpaceWire DMA tests
  - AES-256 self-test
  - CoreMark bechmark
    - Result reduced to 1.44 CoreMark/MHz due to pipelined interconnect
- All tests executed correctly except of timecode test (probably small bug in IP and/or clock domain crossing)

```
[felix@volzotan python]$ ./brave.py
Setting up Coyote brick...

----------------------------------------------------------------
Which test do you want to run?

(a) Memory read/write test
(b) UART test
(c) RISC-V compliance test suite
(d) SpaceWire Timecode test
(e) SpaceWire DMA self-test
(f) SpaceWire DMA external test
(g) AES-256 self-test
(h) CoreMark benchmark
(i) FreeRTOS test

(x) All tests
----------------------------------------------------------------

> Your choice:
```

FreeRTOS port

Had not much experience of how FreeRTOS works internally

- Fortunately, SiFive did a FreeRTOS port for their RISC-V CPUs already
  - SiFive FreeRTOS easily portable to Coyote
  - **Good example for the advantages of an open ISA!**
  - CPU, CSRs (configuration & status registers), timer, and interrupt handling is standardized
  - Therefore, easy to port SW written for another CPU with same ISA (even if much more powerful etc.)

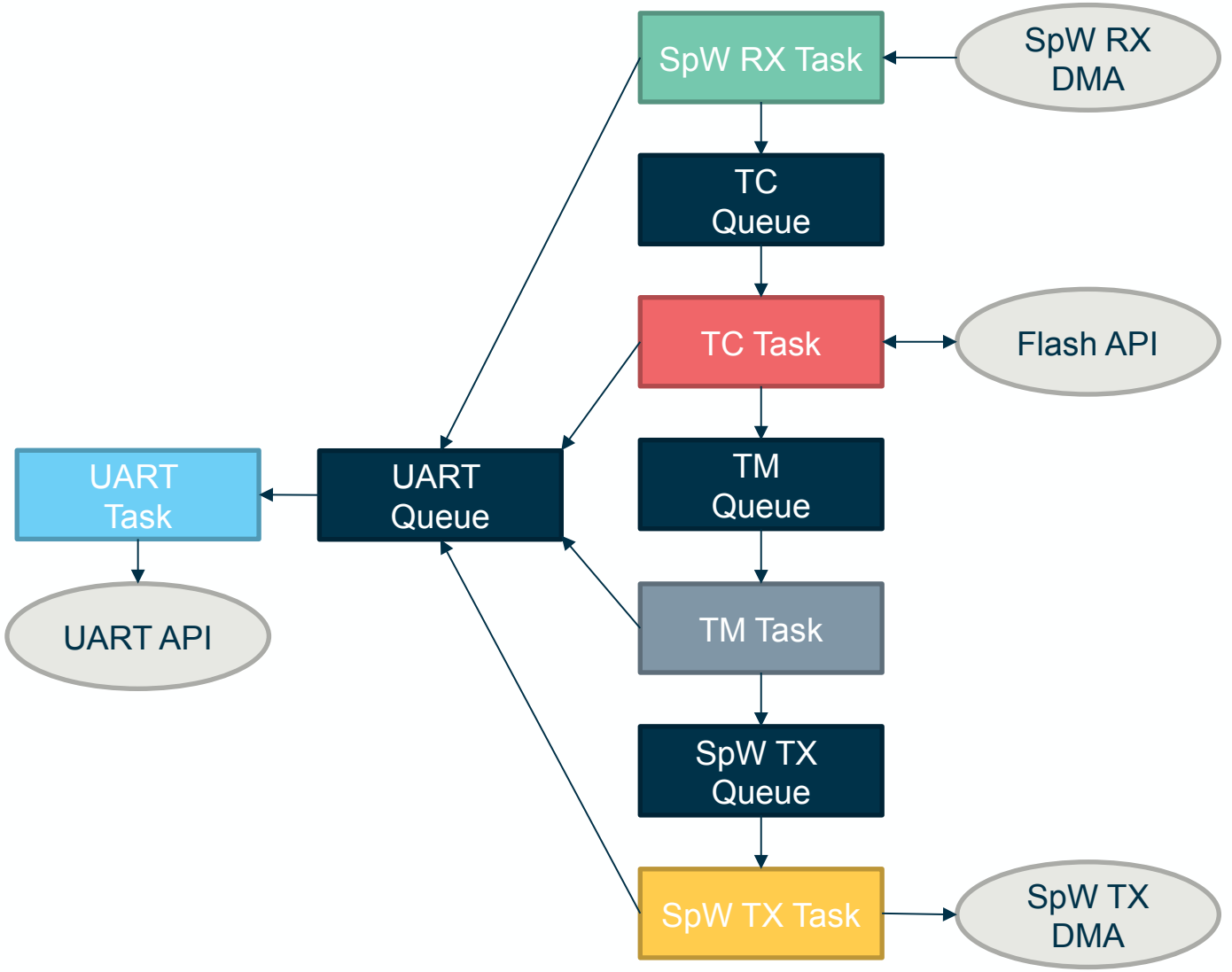
Worked out of the box!

→ Discovered a HW bug in the interrupt handling of the CPU, which has not been much tested before.

- FreeRTOS creates a lot of timer interrupts
- In addition, the application software makes use of SpW DMA interrupts
- Good “stress test” for the interrupt handling

FreeRTOS demo software

- Small example with several tasks
- Time-triggered real time scheduling
- Telecommand and telemetry packets are AES-256 encrypted
- 3 services:
  - Test: Simply relay data
  - Get housekeeping data: Returns HW status and some SW counters
  - SPI flash management: Read, program page, erase sector
- Python script for testing



Welcome

Welcome to RISC-V in Space

and

Stage open for our speakers

- Practical announcements
  - Please bring coffee cups to the bin after use
  - Participants on the webex: questions in chat
  - 25 min time slot = 20 min speech to allow for Q&A

→ THE EUROPEAN SPACE AGENCY
ESA UNCLASSIFIED – For Official Use